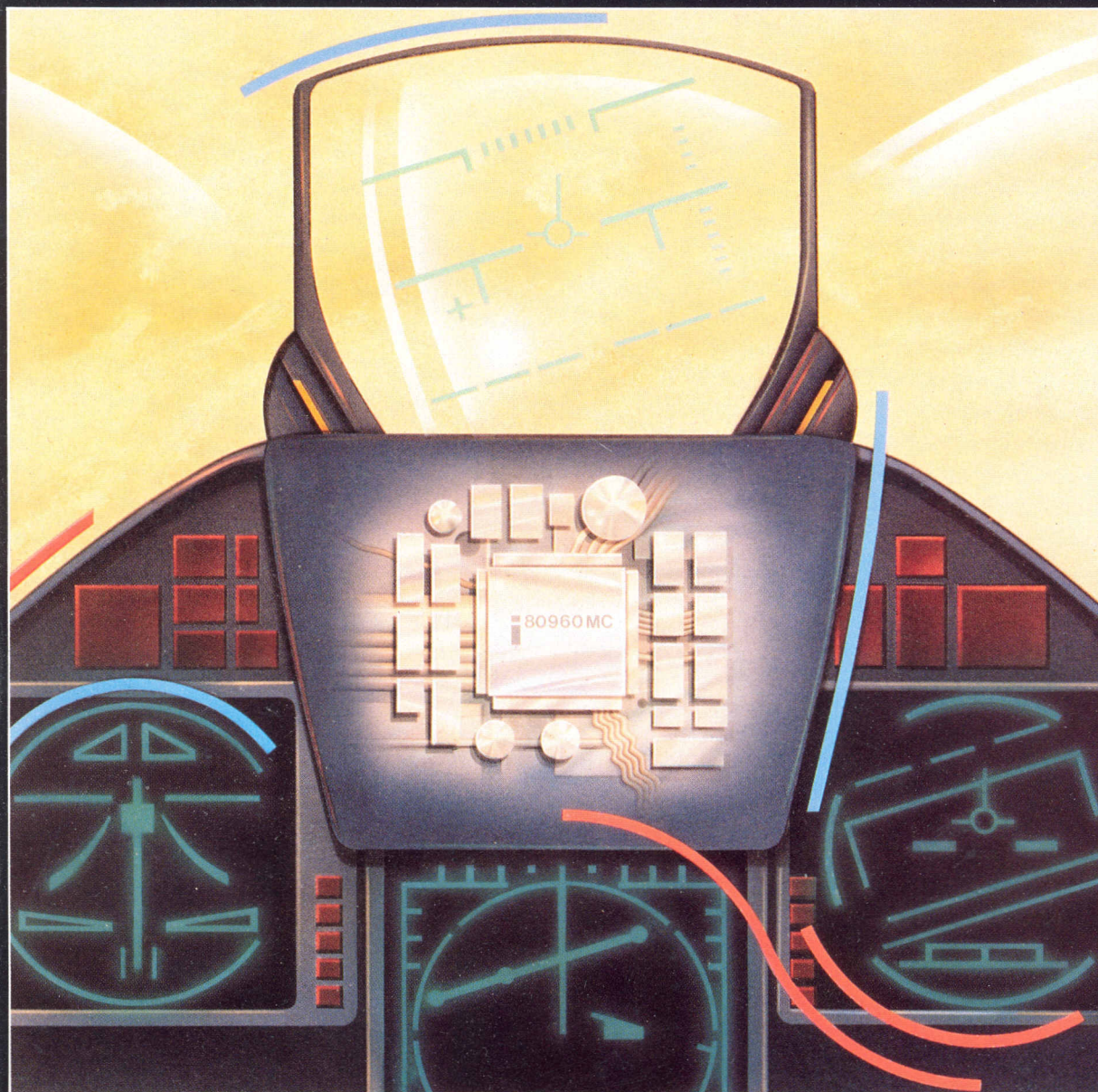




80960MC Hardware Designer's Reference Manual



Order Number: 271079-001



LITERATURE

To order Intel literature write or call:

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Toll Free Number:
(800) 548-4725*

Use the order blank on the facing page or call our Toll Free Number listed above to order literature. Remember to add your local sales tax and a 10% postage charge for U.S. and Canada customers, 20% for customers outside the U.S. Prices are subject to change.

1988 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

NAME	ORDER NUMBER	**PRICE IN U.S. DOLLARS
COMPLETE SET OF 8 HANDBOOKS Save \$50.00 off the retail price of \$175.00	231003	\$125.00
AUTOMOTIVE HANDBOOK (Not included in handbook Set)	231792	\$20.00
COMPONENTS QUALITY/RELIABILITY HANDBOOK (Available in July)	210997	\$20.00
EMBEDDED CONTROLLER HANDBOOK (2 Volume Set)	210918	\$23.00
MEMORY COMPONENTS HANDBOOK	210830	\$18.00
MICROCOMMUNICATIONS HANDBOOK	231658	\$22.00
MICROPROCESSOR AND PERIPHERAL HANDBOOK (2 Volume Set)	230843	\$25.00
MILITARY HANDBOOK (Not included in handbook Set)	210461	\$18.00
OEM BOARDS AND SYSTEMS HANDBOOK	280407	\$18.00
PROGRAMMABLE LOGIC HANDBOOK	296083	\$18.00
SYSTEMS QUALITY/RELIABILITY HANDBOOK	231762	\$20.00
PRODUCT GUIDE Overview of Intel's complete product lines	210846	N/C
DEVELOPMENT TOOLS CATALOG	280199	N/C
INTEL PACKAGING OUTLINES AND DIMENSIONS Packaging types, number of leads, etc.	231369	N/C
LITERATURE PRICE LIST List of Intel Literature	210620	N/C

*Good in the U.S. and Canada

**These prices are for the U.S. and Canada only. In Europe and other international locations, please contact your local Intel Sales Office or Distributor for literature prices.



Intel retains the right to make changes to these specifications at any time, without notice.

80960MC

HARDWARE DESIGNER'S

80960MC HARDWARE DESIGNER'S REFERENCE MANUAL

REFERENCE M

1988



Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

Above, BITBUS, COMMputer, CREDIT, Data Pipeline, FASTPATH, GENIUS, i, i, ICE, ICEL, iCS, iDBP, iDIS, i²ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int_el, int_elBOS, Intel Certified, Intelelevision, intelligent Identifier, intelligent Programming, Inteltec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAP-NET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PC-BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, SupportNET, UPI, and VLSiCEL, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix, 4-SITE.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Distribution
Mail Stop SC6-59
3065 Bowers Avenue
Santa Clara, CA 95051

PART I

SINGLE PROCESSOR HARDWARE DESIGN

CHAPTER 1

INTRODUCTION TO THE 80960MC MICROPROCESSOR

Architectural Attributes for Embedded computing	1-1
Load/Store Design	1-1
Large General Purpose Register Sets	1-2
Small Number of Addressing Modes	1-2
Simplified Instruction Format	1-3
Overlapped Execution	1-3
Minimum cycle operation	1-4
Additional 80960MC Architectural Enhancements	1-4
Floating-point Operation	1-4
Debug Capabilities	1-4
Multitasking Programs	1-5
Memory Management	1-5
Interprocess Communication	1-5
Multiple Processors	1-5
Standard Bus Interface	1-6
Inter-Agent Communication/Coprocessor Capabilities	1-6
Summary	1-6

CHAPTER 2

80960MC SYSTEM ARCHITECTURE

Overview of a Single Processor System Architecture	2-1
80960MC Processor and the L-Bus	2-1
Memory Module	2-2
I/O Module	2-3
Summary	2-3

CHAPTER 3

THE 80960MC PROCESSOR AND THE LOCAL BUS

Overview of the 80960MC L-Bus	3-1
Basic L-Bus States	3-1
L-Bus Signal Groups	3-3
Address/Data	3-3
Control	3-4
L-Bus Transactions	3-6
Clock Signal	3-6
Read Transaction	3-6
Write Transaction	3-9
Burst Transactions	3-10
Timing Generation	3-12
Clock Generation	3-12
Arbitration	3-13
Single 80960MC Processor on the L-Bus	3-15
State Diagram	3-15

Arbitration Timing	3-16
Two 80960MC Processors on the L-Bus	3-17
Bus states for Two 80960MC Processors	3-18
Arbitration Timing for Two 80960MC Processors on the L-Bus	3-18
Bus Exchange Example Between Two 80960MC Processors	3-20
A Peripheral Device As the Default Bus Master	3-21
Inter-Agent Communication (IAC)	3-22
Overview of IAC Operations	3-22
IAC Messages	3-22
Hardware Requirements for IAC Messages	3-22
Message Buffers	3-22
IAC Pin Logic	3-23
External Priority Register	3-24
External Priority and IAC Messages	3-24
Interrupts	3-24
Interrupt Signals	3-25
Interrupt Control Register	3-26
Using the Four Direct Interrupt Pins	3-26
Using an External Interrupt Controller	3-27
Using IAC Requests for Interrupts	3-28
Synchronization	3-28
RESET and Initialization	3-29
RESET Timing Requirements	3-29
RESET Timing Generation	3-29
Initialization	3-30
Error Signals	3-32
Summary	3-33

CHAPTER 4

MEMORY INTERFACE

Basic Memory Interface	4-1
Data Transceivers	4-1
Address Latch/Demultiplexer	4-3
Address Decoder	4-3
Burst Logic	4-3
Timing Control Logic	4-5
Byte Enable Latch	4-5
SRAM Interface	4-6
SRAM Interface Logic	4-6
SRAM Timing Considerations	4-7
DRAM Controller	4-10
Address Multiplexer	4-10
Refresh Interval Timer	4-11
Arbiter	4-12
DRAM Timing and Control	4-12
Timing Considerations for the DRAM Controller	4-15
DRAM Interleaving	4-17
Summary	4-17

CHAPTER 5	
I/O INTERFACE	
Interfacing to 8-bit and 16-bit Peripherals	5-1
General System Interface	5-1
Data Transceivers	5-3
Address Latch/Demultiplexer	5-3
Address Decoder	5-3
Timing Control Logic	5-3
I/O Interface Design Examples	5-4
M8259A Programmable Interrupt Controller	5-5
Interface	5-5
Operation	5-7
82586 Local Area Network Coprocessor Example	5-7
Interface	5-10
Operation	5-11
M82786 Graphics Coprocessor Example	5-12
Interface	5-12
Operation	5-14
Summary	5-15
PART II	
MULTIPROCESSOR DESIGN	
CHAPTER 6	
80960MC MULTIPROCESSOR SYSTEM ARCHITECTURE	
Overview of a Multiprocessor System Architecture	6-1
The L-Bus	6-1
The Advanced Processor Bus	6-2
The BXU Component	6-3
The AP-Bus Interface	6-3
L-Bus Interface	6-3
Cache Directory and Control Logic	6-3
Memory Support Logic	6-4
IAC Support Logic	6-4
I/O Prefetch Logic	6-4
Fault Tolerance Support	6-5
Modes of Operation	6-5
System Configurations	6-5
Active Module	6-6
Passive Module	6-6
Active/Passive Module	6-6
System Implications	6-6
Summary	6-7
CHAPTER 7	
ADVANCED PROCESSOR BUS	
AP-Bus Overview	7-1
AP-Bus Topology	7-2
AP-Bus Signal Groups	7-3
Packet	7-3
Transaction Control	7-4

Error	7-4
Synchronization and Initialization	7-4
Bus Signal Summary	7-5
Non-AP-Bus Module Support Signals	7-5
Memory Organization	7-6
Bus Transactions	7-6
Transaction Specification	7-8
Request Packets and Accepted Replies	7-10
Read Data Transfer	7-11
Read Byte and Read Double-Byte	7-13
Write Data Transfer	7-13
Read-Modify-Write	7-15
Refused Reply Packets	7-16
IAC Transactions	7-17
IAC Flow	7-17
IAC Address Formats	7-17
IAC Message (IAC Type 0011 _B)	7-18
Register Request Using a Logical Address (IAC Type 0010 _B)	7-19
Register Request Using a Physical Address (IAC Type 0100 _B)	7-20
Register Request From the L-Bus (IAC Type 0000 _B)	7-21
Identify Device Order (IAC Type 0111 _B)	7-22
Summary of IAC Transactions	7-23
AP-Bus Protocol	7-24
Arbitration	7-25
Arbitration Process	7-25
Arbitration Example	7-26
Reply Ordering	7-31
Bus Sequencing	7-31
AP-Bus Signal Timing	7-32
General AP-Bus Timing	7-33
AP-Bus Timing Considerations	7-34
Summary	7-34
CHAPTER 8	
AP-BUS INTERFACE USING THE BXU	
BXu Functional Overview	8-1
Major Logical Units	8-2
Modes of Operations	8-2
Processor Mode	8-2
Memory Mode	8-2
Register and Command Summary	8-4
AP-Bus Interface	8-4
Memory Address Recognition	8-4
Guidelines for Recognizer Programming	8-7
Bus Interleaving	8-8
L-Bus Interface	8-9
Memory Address Recognition	8-9
INIT-RAM Recognition	8-9
Private Memory Recognition	8-10
L-Bus and AP-Bus Conversions	8-11
Byte and Double Byte Operations	8-11
RMW Operations	8-11

8-8	Write Acknowledge and No-Acknowledgement IAC Replies	8-12
8-8	Bad-Access Replies	8-12
8-8	Partial Write Operations	8-12
8-8	Use of READY During Normal Memory Operations	8-13
8-8	Use of READY During Special Memory Operations	8-13
	The Cache Directory and Control Logic	8-13
8-8	Overview of Cache Memory Operation	8-13
8-8	Basic Structure of the Cache Memory	8-14
8-8	Cache Configuration and Control	8-16
8-8	Directory	8-17
	Operation with Multiple BXUs	8-17
	Coherency	8-17
	SRAM Interface	8-18
8-8	SRAM Support Logic	8-19
8-8	Address Logic	8-19
8-8	SRAM Chip-Select Logic	8-19
8-8	SRAM Interface Signal Timings	8-20
8-8	Cache Fill Sequence	8-23
8-8	Reaction to a Bad-Access Reply	8-24
	Memory Support	8-24
8-8	Arbitration	8-25
8-8	Normal Operation Support	8-25
8-8	RMW Locks	8-25
8-8	Other Memory Support Facilities	8-26
	IAC Support of the BXU	8-26
	IAC Address Recognition	8-26
	Local Register Request to BXUs on the L-Bus (Type 0000 _B)	8-27
	Register Request Using a Logical Address (Type 0010 _B)	8-28
	Register Request Using a Physical Address (Type 0100 _B)	8-28
	Identify Device Order (Type 0111 _B)	8-29
	IAC Message (Type 0011 _B)	8-29
	IAC Message Support	8-30
8-8	Normal Operation	8-30
8-8	Interaction with the Registers	8-31
	I/O Prefetch Logic	8-32
8-8	Signal Definition	8-33
8-8	Registers and Commands of the I/O Prefetch Unit	8-33
8-8	Start-Channel Command	8-34
8-8	Prefetch Data Buffers	8-35
8-8	Normal Operation	8-35
8-8	Setup	8-35
8-8	Startup	8-35
8-8	Data Transfer	8-35
	Diagnostic Support Functions	8-36
	Cache Testing	8-36
	External SRAM	8-36
	BXU Directory Logic	8-37
	L-Bus Interface Testing	8-37
	BXU Timing	8-37
	Memory Requests	8-38
8-8	Outbound Read Requests	8-38
8-8	Outbound Write Requests	8-39

Outbound RMW-Read Requests	8-39
Outbound RMW-Write Requests	8-40
Access Time for Cacheable Requests	8-40
IAC Requests	8-41
I/O Prefetch Request	8-41
System Configurations	8-42
Active Module	8-43
Passive Module	8-44
Active/Passive Module	8-44
Summary	8-45
CHAPTER 9	
MEMORY AND I/O INTERFACE USING THE BXU	
Basic Memory Interface	9-1
I/O Interface	9-2
General I/O Interface for a Passive Module	9-3
I/O Interface for an Active/Passive Module	9-4
Overview of an Active/Passive Module with the M8259A	9-5
The IAC Generator	9-6
Interconnection	9-6
Timing	9-7
IAC Generator Design	9-8
State Diagram	9-9
Summary	9-10
PART III	
FAULT-TOLERANT SYSTEM DESIGN	
CHAPTER 10	
FUNDAMENTAL CONCEPTS OF FAULT HANDLING	
A Model for Fault Handling	10-1
80960 Fault Handling Approach	10-3
Fault Tolerance Implementation for a 80960 System	10-5
VLSI Replication	10-5
Confinement and Detection	10-6
The Reporting and Recovery Cycle	10-7
Fault Recovery Configuration Examples	10-9
Basic Fault Recovery	10-9
Self-Healing and Continuous Operation Example	10-9
Latent Faults	10-12
Scope of 80960 Fault-Tolerant System Design	10-13
Summary	10-13
CHAPTER 11	
CONFINEMENT AREAS/DETECTION MECHANISMS	
Confinement Areas	11-1
The AP-Bus Confinement Area	11-2
Parity	11-2
Signal Duplication	11-2
Bus Time-Outs	11-2
Module Confinement Area	11-3

Functional Redundancy Checking	11-3
Physical Connection	11-3
FRC Operation and Setup	11-6
Example of Operation Using Confinement Areas	11-6
Summary	11-8
CHAPTER 12	
ERROR REPORTING	
Topology of the Reporting Network	12-1
Error Reporting Protocol	12-2
BERL ₀ -BERL ₀ Timing	12-2
Phase One of the Error Reporting Sequence	12-5
Phase Two of the Error Reporting Sequence	12-7
Error Message Format	12-8
Types of Error Reports	12-10
Error Priorities	12-10
Error Types for Different Detection Mechanisms	12-13
Using Commands to Generate Error Reports	12-13
Error Report Log	12-14
Error Reporting Diagnostics	12-15
BERL ₀ -BERL ₀ Error Detection	12-15
Fault Tolerant Logic	12-16
Control and Accessibility	12-16
Parity Logic	12-16
FRC Logic	12-16
Bus Time-Out Logic	12-17
Error Reporting Logic	12-17
Handling Errors in the Fault-Tolerant Logic	12-17
Summary	12-19
CHAPTER 13	
RECOVERY	
Types of Errors	13-1
Transient Errors	13-1
Permanent Errors	13-3
Unsafe Error Decision	13-3
Retry Sequence	13-3
Retry Operation	13-3
Special Considerations for the Retry Function	13-4
Completion of Operations	13-4
Multi-Word Outbound Read Requests	13-4
Cache Considerations with Retry	13-5
Permanent Error Decision	13-5
Resource Reconfiguration	13-5
Redundancy	13-6
Processor Module Shadowing	13-6
Marrying Processor Modules Using Another Module	13-7
Processor Module Recovery	13-11
Actions by the Failed Module	13-11
Actions by the Partner Module	13-12
Memory Module	13-12
Response to Error Reports	13-12

Memory Controller Interface	13-13
Restoring Failed Memory Locations	13-14
Failures in Partial Write Operations	13-14
Memory Module Shadowing	13-15
Memory Module Marriage	13-15
Bus Switching	13-16
Bus Recovery	13-18
Actions by the Failed Bus	13-18
Actions by the Back-up Bus	13-19
Requests to a Failed Bus	13-20
IAC Operations After a Bus Switch	13-20
Attach-Bus Command	13-21
Communication Between Buses	13-22
Memory Range Recognition Considerations	13-23
Memory Considerations	13-25
Cache Considerations	13-25
I/O Prefetch Considerations	13-26
FRC Splitting	13-26
Software Considerations	13-28
Summary	13-29
CHAPTER 14	
INITIALIZATION	
BXU Initialization	14-1
Default Initialization Phase	14-1
Clock Phase Synchronization and RESET Timing	14-2
Pins Sampled During Default Initialization	14-3
AP-Bus Parameters	14-3
L-Bus and Module Parameters	14-3
Loading Parameters	14-4
BXU State at the End of RESET	14-4
Identification Phase	14-5
Parameterization Phase	14-7
Special COM Protocol	14-7
Cold and Warm Reset Distinction	14-9
Module Shadowing	14-11
Marrying a Primary/Shadow Pair Using a Special Agent	14-11
Special Agent Overview	14-12
QMR Initialization Example	14-17
Local BXU Parameterization	14-18
Identification Phase and Primary/Shadow Marriage	14-20
BXU Identification Assignment	14-20
Initializing the BXUs in the I/O Primary/Shadow Units	14-22
Initializing the I/O Primary-Elect BXU on AP-Bus ₀	14-22
Initializing the I/O Shadow-Elect BXU on AP-Bus ₀	14-23
Initializing the I/O Primary-Elect and Shadow-Elect BXUs on AP-Bus ₀	14-24
Initializing the I/O Primary-Elect BXU on AP-Bus ₁	14-25
Initializing the I/O Shadow-Elect BXU on AP-Bus ₁	14-26
Initializing the I/O Primary-Elect and Shadow-Elect BXUs on AP-Bus ₁	14-27
Processor Primary-Elect and Shadow-Elect Marriage	14-27
Register Summary of the BXUs in the Sample Configuration	14-31
Register Summary of PP ₀	14-34

Register Summary of I/O ₀	14-37
Register Summary of I/O ₀	14-39
Register Summary of PP ₁	14-41
Register Summary of PS ₁	14-43
Register Summary of I/O ₁	14-45
Register Summary of I/O ₁	14-47
Summary	14-49
CHAPTER 15	
FAULT-TOLERANT I/O CONSIDERATIONS	
Overview	15-1
Fault-Tolerant I/O System	15-2
FRC Pair	15-2
Passive Interface Circuit	15-2
Synchronizer	15-3
I/O System	15-4
Comparators	15-5
Design Issues	15-5
QMR I/O	15-6
Handling Interrupts in Fault-Tolerant Systems	15-6
Summary	15-9
APPENDIX A	
8BXU REGISTERS AND COMMANDS	
Descriptions of the Registers and Commands	A-8
FIGURES	
1-1. Local Register Set	1-2
1-2. Global Register Set	1-3
2-1. Basic 80960MC System Configuration	2-2
3-1. Basic L-Bus States	3-2
3-2. L-Bus Signal Groups	3-3
3-3. Byte Enable Timing Diagram	3-5
3-4. Clock Relationships	3-8
3-5. 80960MC Processor Read Transaction	3-8
3-6. 80960MC Processor Write Transaction	3-10
3-7. 80960MC Processor Burst Read Transaction	3-11
3-8. 80960MC Processor Burst Write Transaction	3-12
3-9. Clock Generation Circuit	3-13
3-10. Clock Timing Waveforms	3-14
3-11. L-Bus States with Arbitration	3-16
3-12. Arbitration Timing Diagram for a Bus Master	3-17
3-13. Arbitration Connection Between Two 80960MC Processors	3-18
3-14. L-Bus States for Secondary Bus Master	3-19
3-15. Arbitration Timing Diagram for an SBM	3-20
3-16. Example of a Bus Exchange Transaction	3-20
3-17. Forced Relinquishment Timing Diagram for an SBM	3-21
3-18. Example Flow Chart for an IAC Operation	3-23
3-19. Data Settings	3-23
3-20. Physical Address Interpretation for IAC Messages	3-24
3-21. Interrupt Control Register	3-26

3-22. Timing Diagram for Interrupt Acknowledge Transaction	3-27
3-23. RESET Timing Diagram	3-28
3-24. Asynchronous RESET Circuit	3-29
3-25. Diagram for RESET Timing Generation	3-29
3-26. Synchronous RESET Circuit	3-30
3-27. Initialization Flow Chart	3-31
3-28. RESET Signal Timing Relationship	3-32
4-1. Simplified Block Diagram for Memory Interface Logic	4-2
4-2. Burst Logic Flow Chart	4-4
4-3. Memory Timing Control Block Diagram	4-5
4-4. Logic Diagram for SRAM Interface	4-7
4-5. Critical Timing Path for SRAM Read Operation	4-8
4-6. Critical Timing Path for SRAM Write Transaction	4-9
4-7. DRAM Controller Block Diagram	4-11
4-8. Flow Chart for DRAM Timing and Control Logic	4-13
4-9. Timing Diagram for Two-word DRAM Read Transaction	4-15
4-10. Timing Diagram for Two-word DRAM Write Transaction	4-16
5-1. Simplified I/O Interface	5-2
5-2. I/O Timing Control Block Diagram	5-4
5-3. Block Diagram for M8259A Interface	5-6
5-4. LAN Station	5-8
5-5. Block Diagram for LAN Controller Interface	5-9
5-6. Byte Enable Generation Circuit	5-10
5-7. Operational Flow Diagram for M82586 Interface	5-11
5-8. Block Diagram for M82786 Interface	5-14
5-9. Operational Flow Diagram for M82786 Interface Circuit	5-15
6-1. Basic 80960MC System Configuration	6-1
6-2. Types of Modules	6-5
7-1. AP-Bus Topology	7-2
7-2. AP-Bus Architecture	7-3
7-3. Request Packet Organization	7-7
7-4. Reply Packet Organization	7-8
7-5. IAC Address Format	7-18
7-6. Address Format for IAC Message Transaction	7-19
7-7. Address Format to Access a Register Using a Logical Address	7-20
7-8. Address Format to Access a Register Using a Physical Address	7-21
7-9. Address Format to Access a Register From the L-Bus	7-22
7-10. Address Format to Identify a Device	7-23
7-11. AP-Bus Protocol	7-25
7-12. Arbitration Example	7-28
7-13. Bus Sequencing	7-32
7-14. AP-Bus Signal Timing	7-33
7-15. AP-Bus Timing for Group One Signals	7-34
7-16. AP-Bus Timing for Group Two Signals	7-34
8-1. Address Recognizer Functional Diagram	8-7
8-2. Address Recognizer Example	8-8
8-3. An Example of a Cache Directory Configuration	8-15
8-4. External Logic for Cache Interface Example	8-20
8-5. Signal Timing for 3/6 Read Access Timing (Fast Read Mode)	8-21
8-6. Signal Timing for 4/10 Read Access Timing (Slow Read Mode)	8-21
8-7. Signal Timing for 3/6 Write Access Timing (Fast Write Mode)	8-22
8-8. Signal Timing for 4/10 Write Access Timing (Slow Write Mode)	8-22

8-9. Cache Fill 3/6 Access Timing	8-23
8-10. Cache Fill 3/6 Access Timing (Continued)	8-24
8-11. IAC Address Match Conditions for IAC Type 0000 _B	8-27
8-12. IAC Address Match Conditions for IAC Type 0010 _B	8-28
8-13. IAC Address Match Conditions for IAC Type 0100 _B	8-29
8-14. IAC Address Match Conditions for IAC Type 0111 _B	8-29
8-15. IAC Address Match Conditions for IAC Type 0011 _B	8-30
8-16. Typical Read Timing Diagram	8-38
8-17. Typical Write or Cache Write (Hit or Miss) Timing Diagram	8-39
8-18. Typical RMW-Read Timing Diagram	8-40
8-19. Typical Cache Read Miss Timing Diagram	8-41
8-20. I/O Prefetch Read with Buffer Fill Timing Diagram	8-42
8-21. Types of Modules	8-43
9-1. Memory Interface Logic to the BXU	9-1
9-2. Simplified Block Diagram for Memory Interface Logic to the BXU	9-2
9-3. Overview of the I/O Interface	9-3
9-4. Simplified General System Interface	9-4
9-5. Address Format for the Interrupt IAC	9-5
9-6. Data Word for the Interrupt IAC	9-5
9-7. Block Diagram for Interface Circuit	9-7
9-8. Timing Diagram for a Single-Word IAC Message	9-7
9-9. Block Diagram for IAC Generator	9-9
9-10. State Diagram for IAC Generator	9-10
10-1. Fault Handling Model	10-2
10-2. The 80960 Architectures Separation of Hardware and Software Layers	10-4
10-3. VLSI Republican	10-6
10-4. The Reporting and Recovery State Diagram	10-8
10-5. Fault-Tolerant Alternatives	10-10
10-6. Example of a Basic Fault-Tolerant Design	10-11
10-7. Self-Healing Multiprocessor Configuration	10-11
10-8. QMR Configuration	10-12
11-1. Fault Tolerance Confinement Areas	11-1
11-2. Functional Redundancy Checking	11-4
11-3. Dual Arbitration Network for Each AP-Bus	11-5
11-4. Single Arbitration Network for Each AP-Bus	11-6
11-5. Confinement Area Operation	11-7
12-1. Error Reporting Period	12-2
12-2. Timing for the Start of an Error Report	12-3
12-3. Error Report Propagation for Phase One	12-4
12-4. The Timing of the Error Report Propagation	12-5
12-5. Simultaneous Error Reporting	12-6
12-6. Error Report Propagation for Phase Two	12-8
12-7. Error Message Sequencing	12-9
13-1. Recovery Procedure	13-2
13-2. Primary/Shadow Modules with One System Bus	13-7
13-3. Example of Married Processor Modules	13-8
13-4. Memory Modules	13-13
13-5. Primary/Shadow BXUs with Two AP Buses	13-16
13-6. Permanent Bus Error Recovery	13-17
13-7. System Connection to POPQUE and SSBUSY	13-22
13-8. Bus Recovery Illustration	13-23
13-9. System Vulnerability After Reconfiguration	13-28

14-1. RESET Timing	14-2
14-2. How the Identify Device Order Specifies a Particular BXU	14-6
14-3. Data Field for the Identify Device Order	14-6
14-4. COM Pin Protocol	14-8
14-5. COM Register Loading by Using the COM Pin	14-9
14-6. RC Network for VREF	14-10
14-7. Relationship Between VREF and RESET	14-10
14-8. Marrying a Primary/Shadow Pair Using a Special Agent	14-11
14-9. Special Agent Logic	14-13
14-10. Contents of the "Restart-Processor" IAC	14-14
14-11. IAC Generation Logic	14-14
14-12. Modified State Diagram for IAC Generator	14-15
14-13. Configuration for Example	14-17
15-1. Synchronous I/O Interface	15-1
15-2. Asynchronous I/O Interface	15-3
15-3. Synchronization Circuit	15-4
15-4. Synchronization Circuit Example	15-5
15-5. Alternative to a QMR I/O System	15-7
15-6. I/O Example of Soft QMR I/O System	15-8
A-1. AP-Control Register Contents	A-8
A-2. AP-Mask and AP-Match Register Contents	A-10
A-3. Arbitration-ID Register Contents	A-12
A-4. Bus-Error-ID Register Contents	A-15
A-5. Cache-Configuration Register Contents	A-17
A-6. Cache-Test Register Contents	A-21
A-7. COM Register Contents	A-23
A-8. Component-Specifier Register Contents	A-24
A-9. Error-Log Register Contents	A-25
A-10. Error-Record Register Contents	A-27
A-11. FRC Register Contents	A-28
A-12. FRC-Splitting-Control Register Contents	A-30
A-13. FT1 Register Contents	A-32
A-14. FT2 Register Contents	A-34
A-15. LBI-Control Register Contents	A-37
A-16. Local-Bus-Test Register Contents	A-40
A-17. Lock Register Contents	A-42
A-18. Logical-ID Register Contents	A-44
A-19. L-Bus Mask and Match Register Contents	A-45
A-20. Maxtime Register Contents	A-48
A-21. Module-Error-ID Register Contents	A-49
A-22. Physical-ID Register Contents	A-50
A-23. Prefetch-Control Register Contents	A-51
A-24. Private Memory Mask and Match Register Contents	A-54
A-25. Processor-Priority Register Contents	A-55
A-26. QMR Register Contents	A-57
A-27. Spouse-ID Register Contents	A-62
A-28. System-Bus-ID Register Contents	A-64
A-29. Test-Detection Register Contents	A-66
A-30. Test-Type Register Contents	A-69
TABLES	
3-1. SIZE Signal Decoding	3-4

3-2. Byte Enable Signal Decoding	3-5
3-3. Summary of L-Bus Signals	3-7
3-4. Combination of Bus Masters	3-14
4-1. Byte Enable Signal Decoding	4-6
7-1. AP-Bus Signal Summary	7-5
7-2. Specification Encodings for Packets	7-9
7-3. Memory Block Data for Read Example	7-12
7-4. Read-Request Packet	7-12
7-5. Read-Reply Packet	7-12
7-6. Memory Block Data Before Write Operation	7-14
7-7. Write-Request Packet	7-14
7-8. Write-Acknowledge Packet	7-15
7-9. Memory Block Data After Write	7-15
7-10. Summary of M82965 IAC Transactions	7-24
7-11. Value of the Arbitration-ID Register for Agents in the Example	7-27
8-1. Summary of BXU Modes of Operation	8-3
8-2. Summary of BXU Registers and Commands	8-5
8-3. M80960MC Register Map	8-6
8-4. RMW Lock Map	8-26
8-5. Prefetch Signal Decoding	8-33
8-6. I/O Prefetch Unit	8-34
8-7. Memory Request Access Time	8-38
8-8. Access Time for Cacheable Requests	8-40
8-9. IAC Request Access Time	8-41
8-10. Access Restrictions	8-45
9-1. State Functions	9-11
10-1. Exercising Latent Faults	10-13
12-1. Error Types for Detection Mechanisms	12-14
13-1. Bus Recovery Effects on Interleaving	13-24
13-2. Cache State Change Caused by Bus Switch	13-25
13-3. FRC Splitting Control Bits State	13-27
14-1. System-Bus-Identification Assignments	14-4
14-2. Modified State Functions	14-16
14-3. Summary of identification Values of Each BXU on AP-Bus ₀	14-32
14-4. Summary of identification Values of Each BXU on AP-Bus ₁	14-33
A-1. Summary of BXU Registers and Commands	A-2
A-2. M80960 Register Map	A-4
A-3. Interleave-Control Bit Settings for Different Configurations	A-11
A-4. Mapping of ARB3-ARB0 to the Drive Bits	A-13
A-5. Timing Selections	A-18
A-6. BXU Cache Configurations	A-19
A-7. Address Mapping for Different Configurations	A-19
A-8. SRAM Address Lines	A-20
A-9. Interpretation of Error-Count Value	A-26
A-10. Interleave-Control Bit Settings for Different Configurations	A-38
A-11. BXU-Mode Settings	A-38
A-12. RMW Lock Mapping	A-42
A-13. Addresses for Match and Mask Registers	A-45
A-14. Determination of Which BXU Responds to IAC Type 0010 _B	A-59
A-15. Determination of Which BXU Replies to Requests	A-60
A-16. Parity-Test Bits Versus Parity Tree Corrupted	A-67
A-17. Settings for Test-Type Field	A-70

PREFACE

This manual serves as the definitive hardware reference guide for system designs using the 80960MC processor. Hardware designers can use this manual as a guideline for developing microprocessor systems. Readers of this manual should be familiar with the operating principles of microprocessors and with the 80960MC data sheet.

This manual presents the 80960MC system design from a hardware perspective. Other information on the software architecture, instruction set, and programming of the 80960MC processor can be found in the *80960MC CPU Programmer's Reference Manual*.

Together with the *80960MC Hardware Designer's Reference Manual*, these publications provide a complete description of the 80960MC system for hardware and software designers.

MANUAL ORGANIZATION

The manual is divided into three parts. Part I describes a single processor hardware design using the 80960MC processor with the local bus. Part II discusses a multiprocessor design using 80960MC processors and BXUs with the Advance Processor bus (AP-bus). Finally, Part III describes fault-tolerant system design.

There are 15 chapters and an appendix. The first five chapters describe how to build a hardware system using a single 80960MC processor.

- Chapter 1 briefly introduces the 80960MC component architecture.
- Chapter 2 presents an overview of the 80960MC hardware system design, which includes a system configuration illustrating the various components that constitute an 80960MC system.
- Chapter 3 describes the local bus and the interface to the 80960MC processor. This chapter includes detailed signal descriptions and discusses timing generation, arbitration, interrupt handling, and initialization.
- Chapter 4 discusses techniques for designing memory subsystems.
- Chapter 5 presents guidelines on how to interface I/O devices to the local bus.

The next four chapters describe how to build a multiprocessor hardware system using the Advanced Processor bus.

- Chapter 6 provides an overview of a 80960MC multiprocessor system.
- Chapter 7 focuses on the AP-bus. It describes the AP-bus transactions, AP-bus protocol, an AP-bus signal timing.
- Chapter 8 shows how to interface to the AP-bus by using the BXU. This chapter includes a description of the BXU, diagnostic support functions, performance evaluation, and system considerations.

- Chapter 9 presents guidelines on the memory and I/O interface to a BXU.

The final six chapters present guidelines for fault-tolerant designs.

- Chapter 10 presents an overview fault-tolerant design using the 80960MC processor and the BXU.
- Chapter 11 describes confinement areas and detection mechanisms used in 80960 fault-tolerant designs.
- Chapter 12 discusses the error reporting mechanism used in 80960 fault-tolerant designs.
- Chapter 13 explains the recovery mechanism.
- Chapter 14 shows how to initialize a fault-tolerant system and provides a system initialization example.
- Chapter 15 provides guidelines on how to design fault-tolerant I/O subsystems.

Appendix A contains the descriptions of the registers and commands of the BXU.

Wherever appropriate, design examples are included in the chapters. These designs are based upon functional 80960MC boards and systems, and are simplified for ease of understanding. The simplified versions of these designs have not been tested except for the figures that show part numbers.

NOTATION CONVENTIONS

This manual uses the following style conventions.

- Integer numbers are presented in decimal notation unless otherwise indicated by the subscript “H” for hexadecimal or “B” for binary.
- An active low signal is represented by a line over the signal name. For example, READY is an active low signal.
- Names of bits, address and data fields, and registers of the BXU are noted by a serif italics typeface (e.g., the *Access bit*, the *Component-ID field*, or the *Prefetch-Control register*).
- Names of other items that do not refer to registers of the BXU, such as cache parameters, are noted by a sans serif italics typeface (e.g. the *tag*, *address block*, or *way*).
- For the appendix, the name of the particular register or command described is listed on the page headings.

80960MC Microprocessor

1

Introduction to the 60960MC Microprocessor

1

CHAPTER 1

INTRODUCTION TO THE 80960MC MICROPROCESSOR

The 80960MC is the military version of the 80960 family, designed especially for high reliability embedded applications. At an operating frequency of 20 MHz, this high performance processor can sustain an instruction execution rate of seven and one-half million instructions per second (MIPS), and burst rates of 20 MIPS*. The 80960MC processor enhances embedded system performance by integrating special features to eliminate the need for additional peripheral devices and the associated software overhead. For example, the 80960MC processor offers an on-chip floating-point processing unit, a memory management unit with virtual memory addressability, an improved interrupt handling capability, and support for multiple processors, multitasking, debugging, and tracing.

This chapter describes the architectural attributes and enhancements of the 80960MC processor for embedded computing.

ARCHITECTURAL ATTRIBUTES FOR EMBEDDED COMPUTING

For over a decade, Intel has designed a large variety of 8- and 16-bit microcontrollers to fit the needs of embedded applications. Based on this experience, several architectural attributes shared by both microcontrollers and microprocessors, can be implemented that benefit embedded applications and enhance microprocessor performance. Because the 80960MC processor incorporates these attributes (listed below) in its architecture, embedded applications are easy to design, perform well, and get to market fast.

- Simple load/store design
- Large general-purpose register sets
- Boolean and bit-field instructions
- Small number of operations and addressing modes
- Simplified instruction format
- Minimum cycle operation

Load/Store Design

In the 80960 family architecture, operations are register-to-register, with only LOAD and STORE instructions accessing memory. This attribute simplifies the instruction set and shortens cycle time.

The 80960MC processor uses LOAD and STORE instructions to access memory. It further minimizes accesses to memory by providing a 512-byte, direct-mapped instruction cache. When a memory access is required, the processor can perform a burst transaction that accesses up to four data words with one word transferred every clock cycle.

* DEC VAX 11/780 equals 1 MIP.

Large General-Purpose Register Sets

Because the instructions operate on operands within registers, the 80960 family uses many registers. The 80960MC processor features large, versatile register sets. For maximum flexibility, each processor provides thirty-two 32-bit registers and four 80-bit floating-point registers.

There are two types of general-purpose registers: local and global. The processor automatically accesses the 16 local registers when a procedure call is performed. Multiple sets of local registers are stored on-chip to further increase the efficiency of this register set, as shown in Figure 1-1. The register cache holds up to four local register frames, which means that up to three procedure calls can be made without having to access the procedure stack resident in memory.

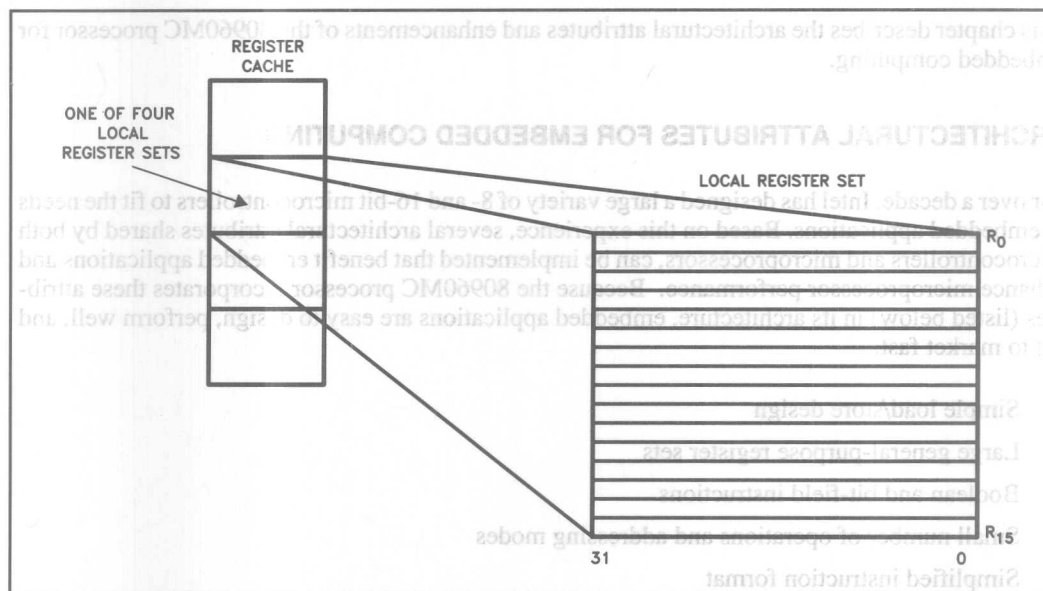


Figure 1-1: Local Register Set

The 20 global registers retain their contents across procedure boundaries. The global registers consist of sixteen 32-bit registers (G_{15} through G_0) and four 80-bit registers (FP_3 through FP_0), as shown in Figure 1-2. While all registers can be used for floating-point operations, the 80-bit registers are used for accumulation of extended precision results.

Small Number of Addressing Modes

The 80960 family uses relatively few addressing modes to facilitate a fast, simple interpretation by the control engine. The 80960MC processor provides simple, fast addressing modes, as well as a few complex addressing modes to allow optimization for code density.

Simplified Instruction Format

A simplified instruction format eases the hardware decoding of instructions, which again speeds control paths. The 80960MC processor's instruction formats are simple and word aligned; all instructions are one word long except for one class that uses the subsequent word as a 32-bit displacement. To further enhance performance, the instructions do not cross word boundaries. This feature eliminates a pipeline stage (that would have to align instructions) and decreases instruction execution time.

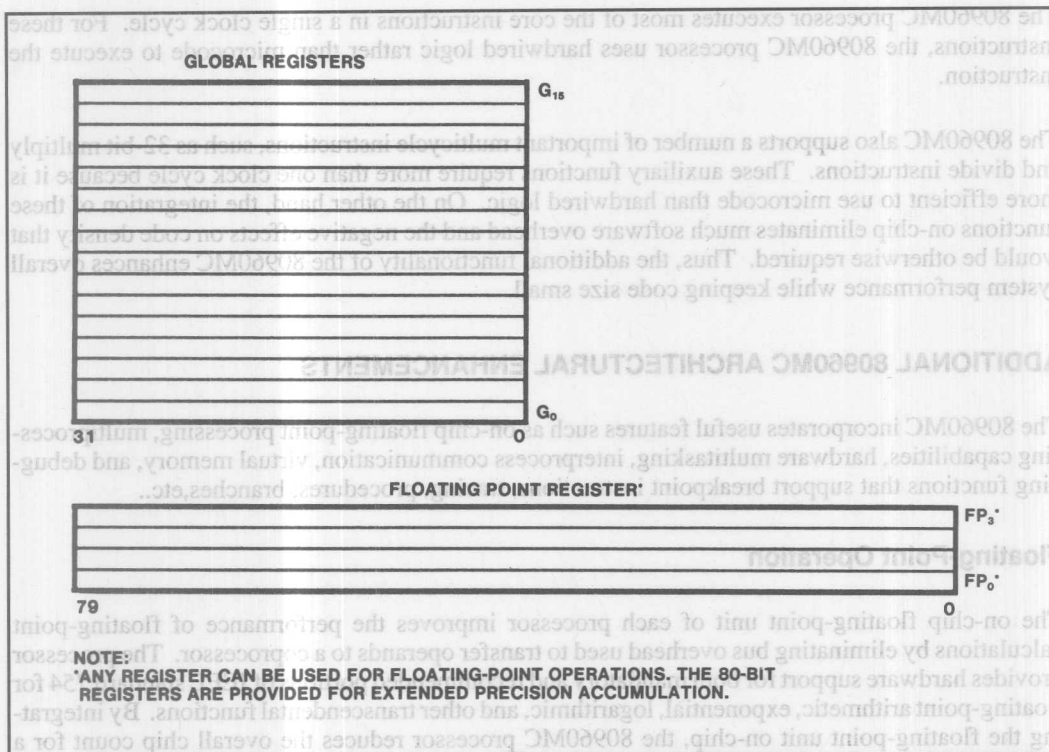


Figure 1-2: Global Register Set

Overlapped Execution

To optimize performance, the 80960MC processor overlaps instruction execution by means of write buffering and register score boarding. Write buffering allows a write instruction to proceed as soon as it is placed in the buffer. It does not have to wait for the actual write operation to occur on the L-bus.

Similarly, register scoreboarding is a design technique that allows the 80960MC to continue execution of instructions when it encounters a **LOAD** instruction. When the **LOAD** instruction

begins, the 80960MC sets a scoreboard bit on the target register. After the target register is loaded with data, the processor resets the bit. While the data is being retrieved, additional instructions that do not reference the target register can be executed. The 80960MC ensures that these additional instructions do not reference the target register by checking the scoreboard bit transparently (no software required). Thus, the scoreboard feature reduces the effect of slow memory speed and provides a useful tool for optimizing procedures.

Minimum Cycle Operation

The 80960MC processor executes most of the core instructions in a single clock cycle. For these instructions, the 80960MC processor uses hardwired logic rather than microcode to execute the instruction.

The 80960MC also supports a number of important multicycle instructions, such as 32-bit multiply and divide instructions. These auxiliary functions require more than one clock cycle because it is more efficient to use microcode than hardwired logic. On the other hand, the integration of these functions on-chip eliminates much software overhead and the negative effects on code density that would be otherwise required. Thus, the additional functionality of the 80960MC enhances overall system performance while keeping code size small.

ADDITIONAL 80960MC ARCHITECTURAL ENHANCEMENTS

The 80960MC incorporates useful features such as on-chip floating-point processing, multiprocessing capabilities, hardware multitasking, interprocess communication, virtual memory, and debugging functions that support breakpoint instructions, tracing, procedures, branches, etc..

Floating-Point Operation

The on-chip floating-point unit of each processor improves the performance of floating-point calculations by eliminating bus overhead used to transfer operands to a coprocessor. The processor provides hardware support for both mandatory and recommended portions of IEEE standard 754 for floating-point arithmetic, exponential, logarithmic, and other transcendental functions. By integrating the floating-point unit on-chip, the 80960MC processor reduces the overall chip count for a system, decreases power consumption, and increases overall performance and reliability.

Debug Capabilities

The processor provides extensive system debug capabilities, an important feature for embedded computing where the ability to instrument an application may be limited. The 80960MC processor allows breakpoint instructions that stop program execution on various events, such as procedure calls, or certain instructions. Another debug facility traces the activity of the processor while it is executing a program. Tracing is done by recording the addresses of instructions that cause trace events to occur. For example, a trace event can occur on the execution of a specific instruction,

branch, or procedure call. To ensure that the 80960MC is operating properly, the processor performs a self-test when it is reset. If the self-test is successful, the 80960MC begins operation, otherwise it enters the stopped state.

Multitasking Programs

The 80960MC processor supports hardware multitasking, interprocess communication, and multiple processor configurations. The 80960MC processor offers several hardware functions designed to support multitasking programs. One unique feature, called self-dispatching, allows a processor to switch itself automatically among scheduled tasks. When self-dispatching is used, the operating system only needs to place the task in a common interprocessor scheduling queue.

Memory Management

For multitasking applications that require software protection and a large address space, the 80960MC processor provides a memory management unit. To ensure the highest level of performance possible and reduce chip count, the memory management unit and translation look-aside buffer are integrated on the chip.

Interprocess Communication

The 80960MC processor supports interprocess communication by using hardware recognized data structures, called communication ports and semaphores. These ports are used to exchange messages and parameters between processes. The 80960MC handles the message passing automatically once the ports are set up by the programmer.

Multiple Processors

The 80960MC processor offers several functions to coordinate the actions of multiple processors. First, the processor can pass messages to each other to initiate actions such as flushing a cache, stopping or starting another processor, or preempting a task. Second, a set of synchronization instructions help maintain the coherency of shared memory. These instructions permit several processors to modify memory at the same time while maintaining data integrity.

The self-dispatching mechanism of the 80960MC previously described provides another means of support for multiple processor applications. This mechanism, in addition to being used in single-processor systems, provides the means to increase the performance of a system by simply adding processors.

Finally, the 80960MC processors synchronize themselves automatically when they perform system operations. A protocol is defined for multiprocessing that provides a low level set of operations. For example, binding a task to a processor means locking the task control block.

STANDARD BUS INTERFACE

The advanced features of the 80960MC processor are implemented using a performance-optimized bus interface. The processor uses a high bandwidth local bus (L-bus) that consists of standard signal groups: a 32-bit multiplexed address/data path and control signals for data transactions. Because of the large amount of caching, the L-bus supports burst transactions that transfer up to four successive data words. Transactions on the L-bus can use 8-, 16-, and 32-bit data types and address up to 4 Giga(G) bytes of physical memory. Bus arbitration can be accomplished by simply using the hold request/hold acknowledge protocol.

INTER-AGENT COMMUNICATION/COPROCESSOR CAPABILITIES

The 80960MC processor offers a flexible way to manage interrupts. It accepts interrupts in one of three ways: by communicating with an external interrupt controller using the standard Interrupt/Interrupt Acknowledge signals, by activating the on-chip interrupt controller, or by accepting an Inter-Agent Communication (IAC) message. This allows the 80960MC to act as a coprocessor on a shared bus with another CPU.

SUMMARY

The 80960MC processor optimizes embedded system performance by using a new 32-bit architecture. The 80960 family architecture includes a load/store design, large general purpose register sets, fast addressing modes, a simplified instruction format, and minimized instruction execution cycles.

To further enhance system performance, the 80960MC processor provides floating-point operation, interrupt controller capabilities, debug functions, multitasking, memory management, interprocess communication, and multiple processor capability. By integrating these functions on-chip, the 80960MC reduces the power requirements and overall chip count for a system.

As a result of the 80960 architecture, the 80960MC processor provides unprecedented performance. For a speed selection of 16 MHz, it can sustain an instruction execution rate of over six million MIPS and burst rates of 16 MIPS, speeds comparable to that of super minicomputers. The high instruction execution rates are made possible through a innovative design that incorporates an on-chip instruction cache with burst-transfer capability.

80960MC System Architecture **2**

CHAPTER 2

80960MC SYSTEM ARCHITECTURE

This chapter illustrates the flexibility and power of the 80960MC system architecture using the advanced 32-bit 80960MC processor. This chapter examines system configurations from a general perspective to explain the design concepts. Subsequent chapters describe the details of the system design.

OVERVIEW OF A SINGLE PROCESSOR SYSTEM ARCHITECTURE

The central processing module, memory module, and I/O module form the natural boundaries for the hardware system architecture. The modules are connected together by the high bandwidth 32-bit multiplexed L-bus, which can transfer data at a maximum sustained rate of 42 Mega(M) bytes per second for an 80960MC processor operating at 16 MHz.

Figure 2-1 shows a simplified block diagram of a possible system configuration. The heart of this system is the 80960MC processor, which fetches program instructions, executes code, manipulates stored information, and interacts with I/O devices. The high bandwidth L-bus connects the 80960MC processor to memory and I/O modules. The 80960MC processor stores system data and instructions and programs in the memory module. By accessing various peripheral devices in the I/O module, the 80960MC processor directly supports communication with other ancillary subsystems.

80960MC Processor and the L-Bus

The 80960MC processor performs bus operations using multiplexed address and data signals and provides all the necessary control signals. For example, standard Intel control signals are provided, such as Address Latch Enable (\overline{ALE}), Address/Data Status (\overline{ADS}), Write/Read command ($\overline{W/R}$), Data Transmit/Receive ($\overline{DT/R}$), and Data enable (\overline{DEN}). The 80960MC processor also generates byte enable signals that specify which bytes on the 32-bit data lines are valid for the transfer.

The L-bus supports burst transactions, which access up to four data words at a maximum rate of one word per clock cycle. The 80960MC processor uses the two low-order address lines to indicate how many words are to be transferred. The 80960MC processor performs burst transactions to load the on-chip 512-byte instruction cache to minimize memory accesses for instruction fetches. Burst transactions can also be used for data accesses.

To transfer control of the bus to an external bus master, the 80960MC processor provides two arbitration signals: hold request (HOLD) and hold acknowledge (HLDA). After receiving HOLD, the processor grants control of the bus to an external bus master by asserting HLDA.

The 80960MC processor provides a flexible interrupt structure by using an on-chip interrupt controller, an external interrupt controller, or both. The type of interrupt structure is specified by an internal interrupt vector register. For a system with multiple processors, another method is available,

called inter-agent communication (IAC) where a processor can interrupt another processor by sending an IAC message.

Complete details of the L-bus and bus operations are discussed in Chapter 3.

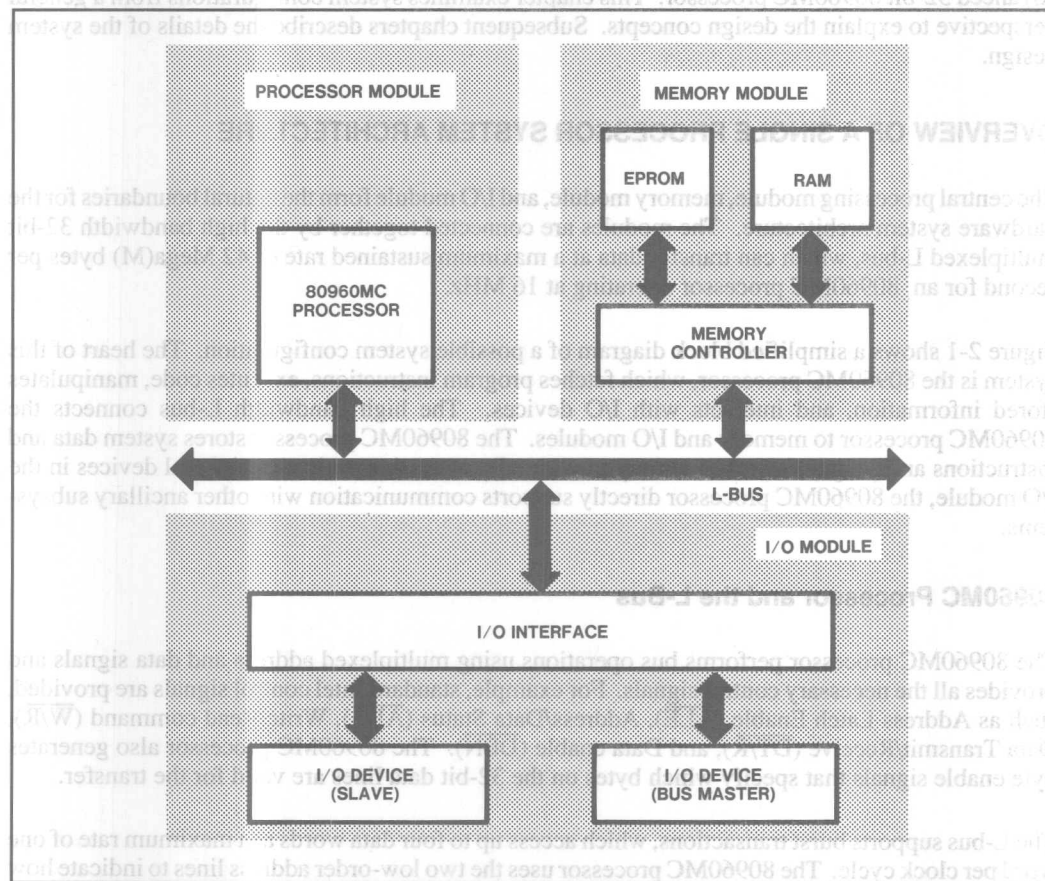


Figure 2-1: Basic 80960MC System Configuration

Memory Module

A memory module can consist of the memory controller, Erasable Programmable Read Only Memory (EPROM), and static or dynamic Random Access Memory (RAM). The memory controller first conditions the L-bus signals for memory operation. It demultiplexes the address and data lines, generates the chip select signals from the address, detects the start of the cycle for burst mode operation, and latches the byte enable signals.

The memory controller generates the control signals for EPROM, SRAM, and DRAM. In particular, it provides the control signals, multiplexed row/column address, and refresh control for dynamic RAMs. The controller can be designed to accommodate the burst transaction of the 80960MC processor by using the static column mode or nibble mode features of the dynamic RAM. In addition to supplying the operation signals, the controller generates the $\overline{\text{READY}}$ signal to indicate that data can be transferred to or from the 80960MC processor.

The 80960MC processor directly addresses up to 4G bytes of physical memory. The processor does not allow burst accesses to cross a 16-byte boundary to ease the design of the controller. Each address specifies a four-byte data word within the block. Individual data bytes can be accessed by using the four byte enable signals from the 80960MC processor.

Chapter 4 provides design guidelines for the memory controller.

I/O Module

The I/O module consists of the I/O components and the interface circuit. I/O components can be used to allow the 80960MC processor to use most of its clock cycles for computational and system management activities. Time consuming tasks can be off-loaded to specialized slave-type components, such as the M8259A Programmable Interrupt Controller. Some tasks may require a master-type component, such as the 82586 Local Area Network Control.

The interface circuit performs several functions. It demultiplexes the address and data lines, generates the chip select signals from the address, produces the I/O read or I/O write command from the processor's W/R signal, latches the byte enable signals, and generates the $\overline{\text{READY}}$ signal. Because these functions are the same as some of the functions of the memory controller, the same logic can be used for both interfaces. For master-type peripherals that operate on a 16-bit data bus, the interface circuit translates the 32-bit data bus to a 16-bit data bus.

The 80960MC processor uses memory-mapped addresses to access I/O devices. This allows the CPU to use many of the same instructions to exchange information for both memory and peripheral devices. Thus, the powerful memory-type instructions can be used to perform 8-, 16-, and 32-bit data transfers.

Chapter 5 describes design guidelines for the I/O interface by examining representative design examples.

SUMMARY

The basic hardware system configuration is modular and flexible. The processor, memory, and I/O modules form the natural boundaries in the basic hardware system architecture. The high-bandwidth L-bus that supports burst transfers is used for the data path between the 80960MC processor and other modules.

This chapter presented an overview for basic hardware system design. The next three chapters discuss the details of the L-bus, memory modules, and I/O modules.

*The 80960MC Processor
and the Local Bus*

3

The 80960MC Processor and the Local Bus

3

CHAPTER 3 THE 80960MC PROCESSOR AND THE LOCAL BUS

The 32-bit multiplexed local bus (L-bus) connects the 80960MC processor to memory and I/O and forms the backbone of any 80960MC processor based system. This high bandwidth bus provides burst-transfer capability allowing up to four successive 32-bit data word transfers at a maximum rate of one word every clock cycle. In addition to the L-bus signals, the 80960MC processor uses other signals to communicate to other bus masters. This chapter, which describes these signals and the associated operations, follows the outline shown below:

- L-bus states and their relationship to each other
- L-bus signal groups, which consist of address/data and control
- L-bus read, write, and burst transactions
- L-bus timing analyses and timing circuit generation
- Related L-bus operations such as arbitration, interrupt, and reset operations

OVERVIEW OF THE 80960MC L-BUS

The L-bus forms the data communication path between the various components in a basic 80960MC hardware system. The 80960MC processor utilizes the L-bus to fetch instructions, to manipulate information from both memory and I/O devices, and to respond to interrupts. To perform these functions at a high data rate, the 80960MC processor provides a burst mode, which transfers up to four data words at a maximum rate of one 32-bit word per clock cycle. The 80960MC L-bus has the following features:

- 32-bit multiplexed address/data path
- High data bandwidth relative to the speed selection of the 80960MC processor
- Four byte enables and a four-word burst capability that allow transfers from 1 to 16 bytes in length
- Support for TTL latches and buffers.

BASIC L-BUS STATES

The L-bus has five basic bus states: idle (T_i), address (T_a), data (T_d), recovery (T_r), and wait (T_w). During system operation, the 80960MC processor continuously enters and exits different bus states as shown in Figure 3-1. This state diagram assumes that only one bus master resides on the L-bus. The local bus occupies the idle (T_i) state when no address/data transfers are in progress. When a new request is received, the L-bus enters the T_a state to transmit the address.

Following a T_a state, the L-bus enters a T_d state to transmit or receive data on the address/data lines provided that the data is ready (indicated by the assertion of $\overline{\text{READY}}$ at the input of the processor).

If the data is not ready, the L-bus enters a T_w state and remains in this state until data is ready. T_w states may be repeated as many times as necessary to allow sufficient time for the memory or I/O device to respond.

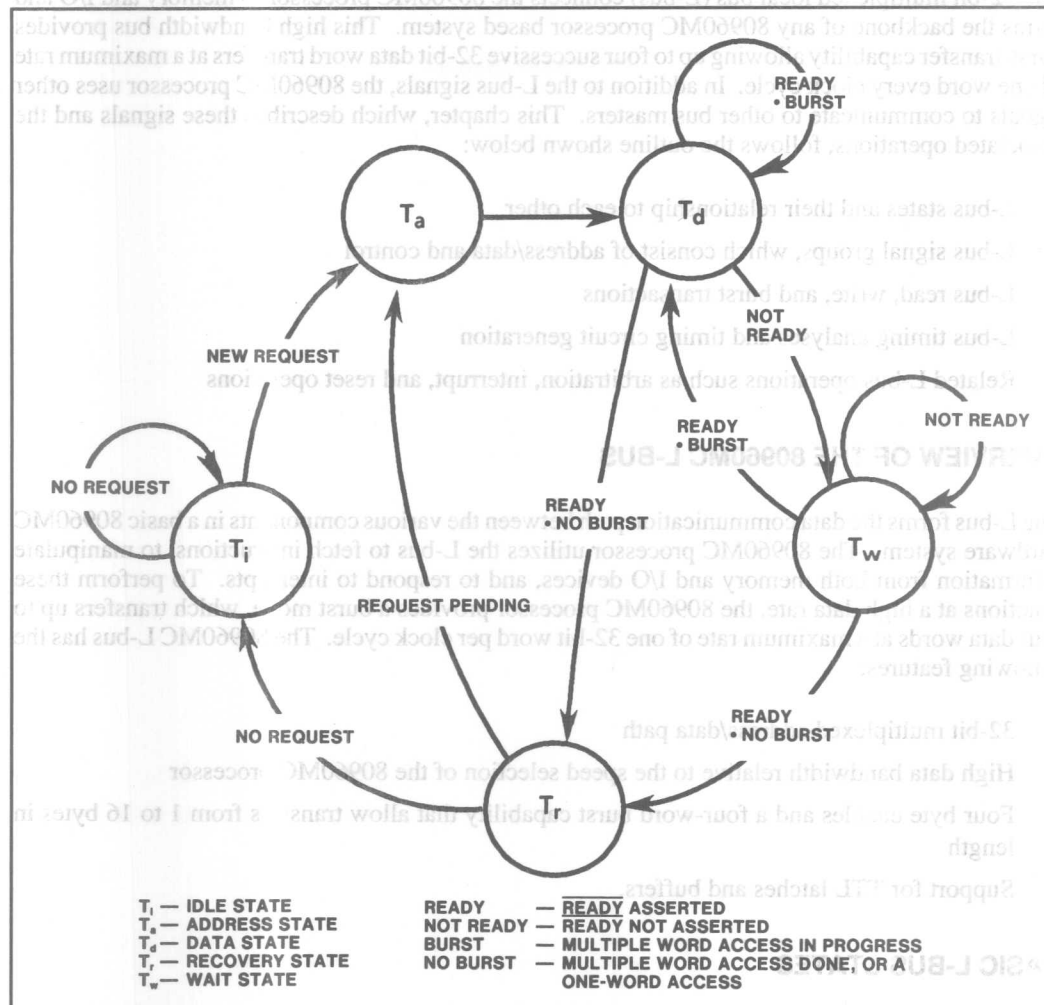


Figure 3-1: Basic L-Bus States

After a data word is transferred in a non-burst transaction, the L-bus exits the T_d or T_w state and enters the recovery (T_r) state. In the case of a burst transaction, the local bus will exit the T_d or T_w state and re-enter the T_d state to transfer the next data word. Once all data words have been transferred in a burst transaction (up to four), the L-bus enters the T_r state to allow devices on the L-bus to recover.

When the recovery state is complete the L-bus will enter the T_i state if no new request is pending. If a request is pending, the L-bus will enter the T_a state to transmit the new address.

L-BUS SIGNAL GROUPS

Signals on the L-bus, shown in Figure 3-2, consist of two basic groups: address/data, and control. A description of both of these signal groups is provided in this section.

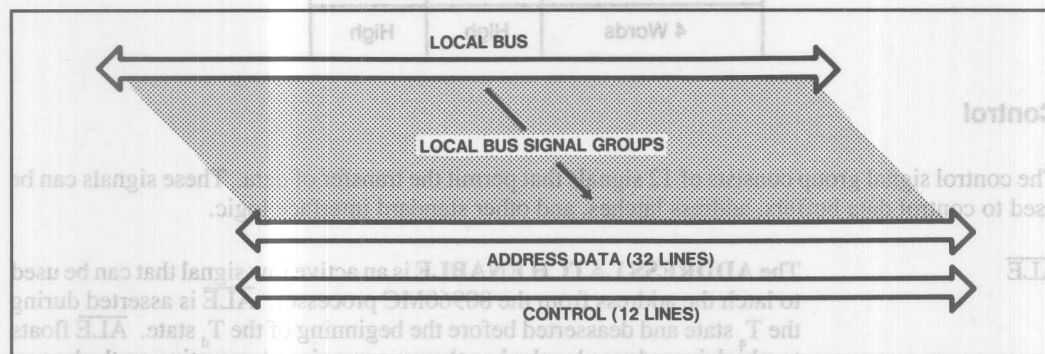


Figure 3-2: L-Bus Signal Groups

Address/Data

The address/data signal group consists of 32 bi-directional lines. These signals are multiplexed to serve a dual purpose depending upon the bus state.

LAD_{31} - LAD_2

LOCAL ADDRESS/DATA₃₁ through LOCAL ADDRESS/DATA₂ represent the address signals on the L-bus during the T_a state. LAD_2 is the least significant bit, and LAD_{31} is the most significant address bit. LAD_{31} through LAD_2 contain a physical word address. LAD_1 and LAD_0 specify the number of data words to transfer within a burst transaction. The address/data signals float to a high impedance state when not activated.

SIZE (LAD_1 - LAD_0)

The **SIZE** signals indicate whether one, two, three, or four words are transferred during the current transaction. These signals are valid during the T_a state of the L-bus. The encoding of the LAD_1 and LAD_0 signals to represent the size of a burst transaction is shown in Table 3-1.

LAD_{31} - LAD_0

LOCAL ADDRESS/DATA₃₁ through LOCAL ADDRESS/DATA₀ represent the data signals on the L-bus during the T_d and T_w states. LAD_0 is the least significant bit, and LAD_{31} is the most significant data bit. The address/data signals float to a high impedance state when not activated.

Table 3-1: SIZE Signal Decoding

Word Selection	LAD ₁	LAD ₀
1 Word	Low	Low
2 Words	Low	High
3 Words	High	Low
4 Words	High	High

Control

The control signal group consists of 12 signals that permit the transfer of data. These signals can be used to control data buffers, address latches, and other standard interface logic.

$\overline{\text{ALE}}$

The **ADDRESS LATCH ENABLE** is an active low signal that can be used to latch the address from the 80960MC processor. $\overline{\text{ALE}}$ is asserted during the T_a state and deasserted before the beginning of the T_d state. $\overline{\text{ALE}}$ floats to a high impedance level when the processor is not operating on the bus, or is at the end of any bus access (i.e., the recovery cycle).

$\overline{\text{ADS}}$

ADDRESS STATUS is an active low signal that is driven by the 80960MC processor to indicate an address state. $\overline{\text{ADS}}$ is asserted during every T_a state and deasserted during the following T_d and T_w states. For a burst transaction, $\overline{\text{ADS}}$ is asserted again every T_d (and T_w) state where **READY** was asserted in the prior cycle. The $\overline{\text{ADS}}$ signal is an open drain output.

$\text{DT}/\overline{\text{R}}$

DATA TRANSMIT/RECEIVE indicates the direction of data flow to or from the 80960MC processor. For a read operation or an interrupt acknowledgement, $\text{DT}/\overline{\text{R}}$ is low during the T_a , T_d , and T_w states to indicate that data flows into the 80960MC processor. For a write operation, $\text{DT}/\overline{\text{R}}$ is high during the T_a , T_d , and T_w states to indicate that data flows from the 80960MC processor. $\text{DT}/\overline{\text{R}}$ never changes states when $\overline{\text{DEN}}$ is asserted. The $\text{DT}/\overline{\text{R}}$ line is an open drain output of the 80960MC processor.

$\overline{\text{DEN}}$

DATE ENABLE is an active-low signal that can be used to enable data transceivers. $\overline{\text{DEN}}$ is asserted during all T_d and T_w states. The $\overline{\text{DEN}}$ line is an open drain output of the 80960MC processor.

$\text{W}/\overline{\text{R}}$

THE WRITE/READ signal instructs a memory or I/O device to write or read data on the L-bus. The 80960MC processor asserts $\text{W}/\overline{\text{R}}$ during a T_a state. The signal remains valid during subsequent T_d and T_w states. $\text{W}/\overline{\text{R}}$ is an open drain output of the 80960MC processor.

THE BYTE ENABLE output signals of the 80960MC processor specify which bytes (up to four) on the 32-bit data bus are transferred during the transaction. Table 3-2 shows the decoding scheme for these signals.

Table 3-2 : Byte Enable Signal Decoding

Byte Enable Signal	Address Line Selection
\overline{BE}_0	LAD ₇ -LAD ₀
\overline{BE}_1	LAD ₁₅ -LAD ₈
\overline{BE}_2	LAD ₂₃ -LAD ₁₆
\overline{BE}_3	LAD ₃₁ -LAD ₂₄

The byte enable signals are valid from the 80960MC processor before data is transferred, as shown in Figure 3-3 (assumes no wait states). The byte enable signals that are valid for the first data word are specified during the T_a state. For a four-word burst transaction, the byte enable signals that are valid for the second word are asserted during the first data state (T_{d0}), for the third word during the second data state (T_{d1}), and for the fourth word during the third data state (T_{d2}). The byte enable signals are undefined during the last data state (T_{d3}) of the last word transferred.

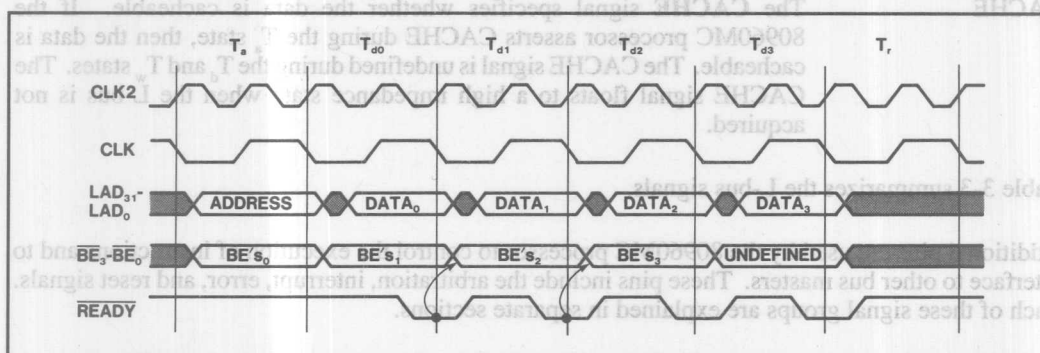


Figure 3-3: Byte Enable Timing Diagram

Although not shown in the diagram, the byte enable signals of each word are latched internally by the 80960MC processor and remain valid during every data or wait state until \overline{READY} is applied. After \overline{READY} is applied the byte enable signals change during the next T_d state or become undefined for the last data transfer.

The 80960MC processor asserts only adjacent byte enables. For example, the 80960MC processor does not perform a bus operation with only \overline{BE}_0 and \overline{BE}_2 active. The Byte Enable lines are open drain outputs.

READY

The **READY** signal indicates that the data on the L-bus can be sampled (read) or removed (write) by the 80960MC processor. If **READY** is not asserted following the T_a state or in between T_d states, a T_w state is generated. **READY** is an active-low input signal to the 80960MC processor.

LOCK

Bus **LOCK** prevents other bus masters from gaining control of the L-bus during a bus operation. It is activated by certain 80960MC processor operations and instructions.

The 80960MC processor uses the bus **LOCK** signal when it performs a RMW memory operation. When the processor performs a RMW-Read operation, it asserts the **LOCK** signal during the T_a state and holds **LOCK** asserted. If the **LOCK** signal was already asserted, the processor waits until this signal is deasserted before performing the RMW-Read operation. The processor deasserts the **LOCK** signal during the T_a state when it performs a RMW-Write operation.

The 80960MC processor also asserts the **LOCK** signal during the interrupt acknowledge sequence. **LOCK** is an input and an open drain output signal from the 80960MC processor.

CACHE

The **CACHE** signal specifies whether the data is cacheable. If the 80960MC processor asserts **CACHE** during the T_a state, then the data is cacheable. The **CACHE** signal is undefined during the T_d and T_w states. The **CACHE** signal floats to a high impedance state when the L-bus is not acquired.

Table 3-3 summarizes the L-bus signals.

Additional pins are used by the 80960MC processor to control the execution of instructions and to interface to other bus masters. These pins include the arbitration, interrupt, error, and reset signals. Each of these signal groups are explained in separate sections.

L-BUS TRANSACTIONS

The 80960MC processor uses the L-bus signals to perform transactions in which data is transferred to (or from) the CPU from (or to) another component. During a transaction, the 80960MC processor can transfer up to four words of data for a single address to enhance system throughput. This is especially useful when loading cache memory.

Table 3-3: Summary Of L-Bus Signals

Signal Group	Signal Symbol	Signal Function	Active State	Direction	Type of Output
Local Address/Data	Address (LAD ₃₁ -LAD ₂)	32-bit address	T _a	O	3-state
	Data (LAD ₃₁ -LAD ₀)	32-bit data	T _d , T _w	I/O	3-state
	Size (LAD ₁ -LAD ₀)	Specifies number of words to transfer	T _a	O	3-state
Control	ALE	Enables address latch	T _a	O	3-state
	ADS	Identifies an address state	T _a , T _d , T _w	O	Open drain
	DT/R	Controls direction of data flow	T _a , T _d , T _w	O	Open drain
	DEN	Enables data transceiver/latch	T _d , T _w	O	Open drain
	W/R	Read/write command	T _a , T _d , T _w	O	Open drain
	BE ₃ -BE ₀	Specifies which data bytes to transfer	T _a , T _d ² , T _w ²	O	Open drain
	READY	Indicates data is ready to transfer	T _d , T _w	I	—
	LOCK	Locks bus	Any	I/O	Open drain
	Cache	Indicates cacheable transaction	T _a	O	3-state

NOTES:

1. Active after the first assertion of READY.
2. Active except for the last transfer.

Clock Signal

The 80960MC hardware system typically uses two clock signals, CLK2 and CLK, to synchronize the transitions between L-bus states. CLK2 is the clock input to the 80960MC and is double the specified processor frequency. CLK is an optional clock signal derived through external logic to provide a convenient reference of L-bus cycles and can be used to drive peripheral devices. CLK is one-half the frequency of CLK2, and is neither an input or output of the 80960MC processor. Figure 3-4 shows the relationship between the system CLK2 and CLK.

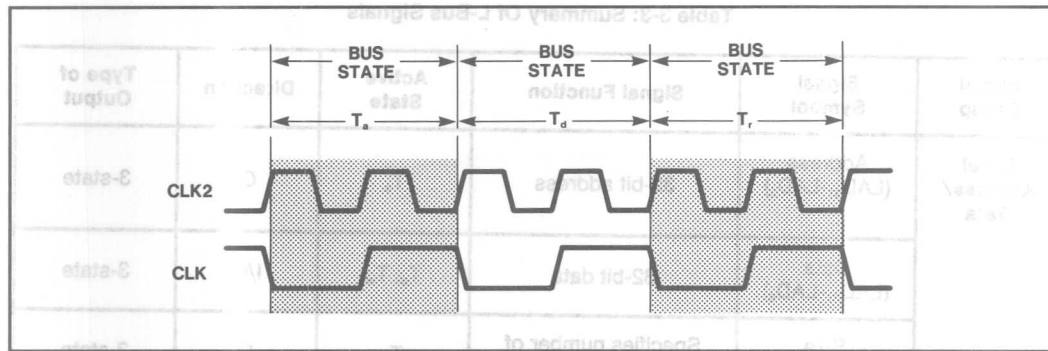


Figure 3-4: Clock Relationships

Read Transaction

Figure 3-5 shows a typical timing diagram for a read transaction (for exact timings, see the 80960MC processor data sheet). The following sequence of events explains the flow of the timing diagram. For simplicity, no wait states are shown.

1. The 80960MC processor generates several signals during the T_a state.
 - It transmits the address on the address/data lines. LAD_1 and LAD_0 specify a single word transaction.
 - It asserts \overline{ALE} .
 - It asserts \overline{ADS} .
 - It asserts \overline{BE}_3 - \overline{BE}_0 to specify which bytes are used when reading the data word.
 - It brings $\overline{W/R}$ low to denote a read operation.
 - It brings $\overline{DT/R}$ low.
2. During the T_d state, several actions occur.
 - The 80960MC processor asserts \overline{DEN} . \overline{DEN} can be used to enable data transceivers. \overline{READY} is asserted by external timing logic and data is transmitted from the storage devices. If \overline{READY} is not asserted, the L-bus will enter a T_w state. The T_w state is repeated, until \overline{READY} is asserted.
 - The 80960MC processor reads the data on the address/data lines.
3. The T_r state follows the data state. This allows the system components adequate time (one processor clock cycle) to remove their outputs from the bus before the 80960MC processor generates the next address on the address/data lines. During the T_r state $\overline{W/R}$, $\overline{DT/R}$, and \overline{DEN} become inactive.

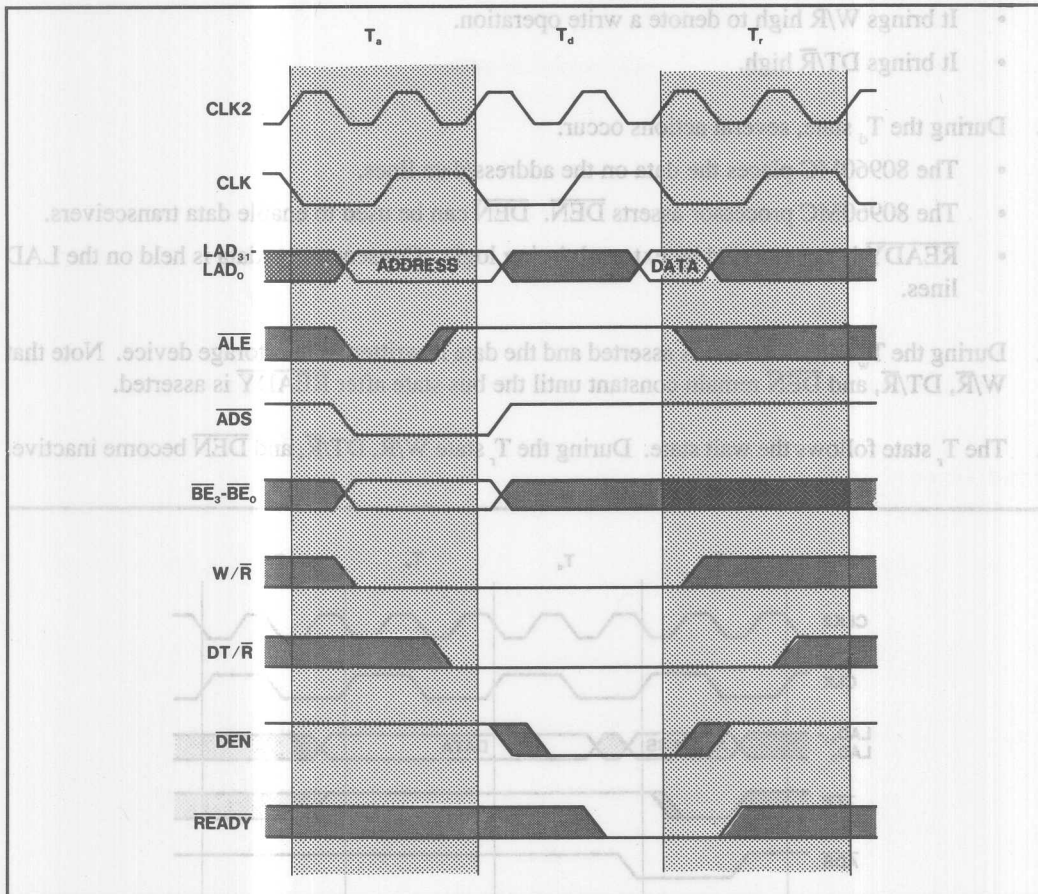


Figure 3-5: 80960MC Processor Read Transaction

Write Transaction

Figure 3-6 shows a typical timing diagram for a write transaction with one wait state. The following sequence of events explains the flow of the timing diagram.

- Similar to the read transaction, the 80960MC processor generates several signals during the T_0 state.
 - It transmits the address on the address/data lines. LAD₁ and LAD₀ specify a single word transaction.
 - It asserts $\overline{\text{ALE}}$.
 - It asserts $\overline{\text{ADS}}$.
 - It asserts $\overline{\text{BE}}_3\text{-}\overline{\text{BE}}_0$ to specify which bytes are used when writing the data word.

- It brings W/\bar{R} high to denote a write operation.
 - It brings DT/\bar{R} high.
2. During the T_d state, several actions occur.
 - The 80960MC places the data on the address/data lines.
 - The 80960MC processor asserts \overline{DEN} . \overline{DEN} can be used to enable data transceivers.
 - \overline{READY} is not asserted by external timing logic. Consequently, data is held on the LAD lines.
 3. During the T_w state \overline{READY} is asserted and the data is written to the storage device. Note that W/\bar{R} , DT/\bar{R} , and \overline{DEN} remain constant until the bus state after \overline{READY} is asserted.
 4. The T_r state follows the wait state. During the T_r state W/\bar{R} , DT/\bar{R} , and \overline{DEN} become inactive.

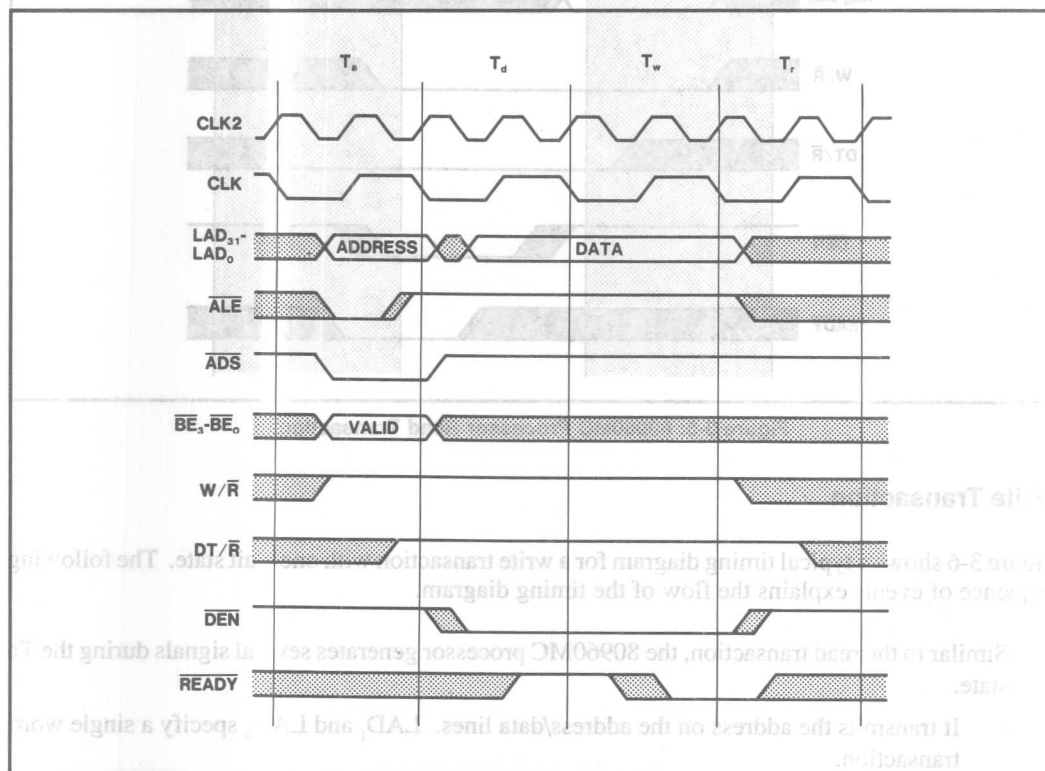


Figure 3-6: 80960MC Processor write Transaction

Burst Transactions

The 80960MC processor supports burst transactions that read or write up to four words (16 contiguous bytes) at a maximum rate of one word every L-bus cycle. The byte enable signals are valid for each word to allow partial-word write operations to contiguous bytes within a word. The CACHE output signal during a T_a state applies to all words of a burst transaction.

A burst read or write transaction is similar to a single word read or write operation. It differs primarily in the number of data words transferred: the basic transaction always transfers one data word, the burst transaction transfers up to four data words. For a burst transaction, the byte enable signals are applied during the T_a state, and subsequently during every T_d or T_w state before the data word is transferred. Figure 3-7 shows the timing for a three-word burst read transaction without wait states. Figure 3-8 shows the timing for a two-word burst write transaction with a wait state occurring during the transfer of the first word. Note that the byte enable signals remain constant until the data state after $READY$ is asserted.

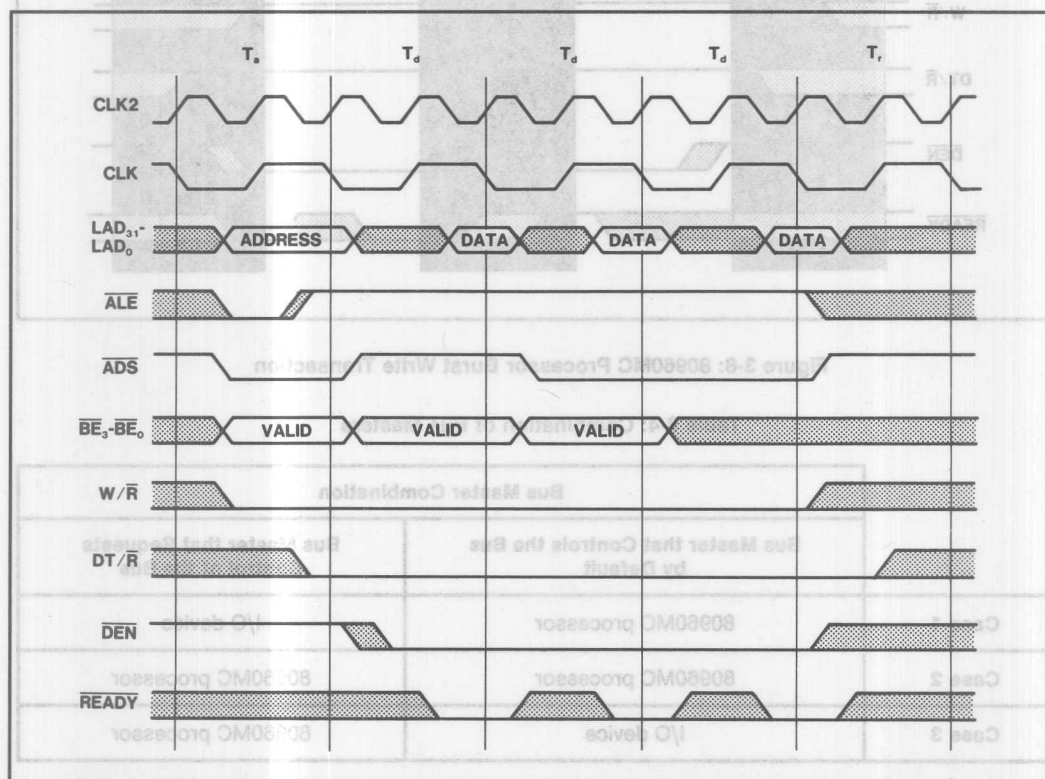


Figure 3-7: 80960MC Processor Burst Read Transaction

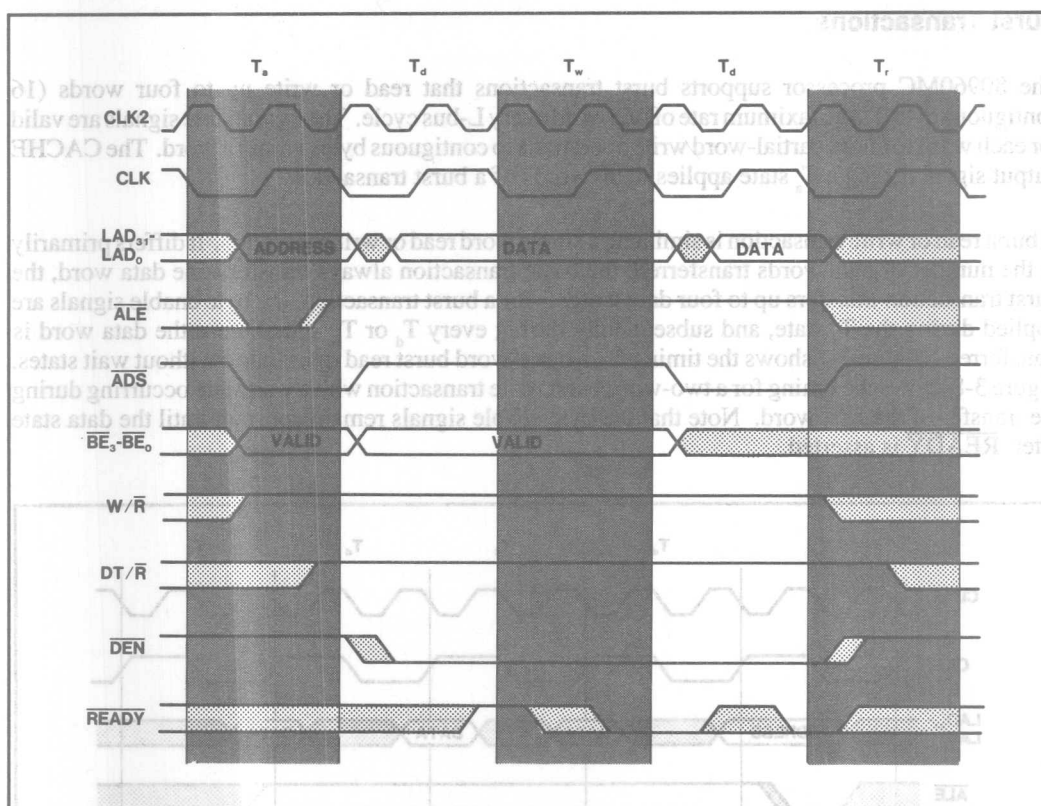


Figure 3-8: 80960MC Processor Burst Write Transaction

Table 3-4: Combination of Bus Masters

	Bus Master Combination	
	Bus Master that Controls the Bus by Default	Bus Master that Requests Control of the Bus
Case 1	80960MC processor	I/O device
Case 2	80960MC processor	80960MC processor
Case 3	I/O device	80960MC processor

TIMING GENERATION

In an 80960MC processor-based system, timing signals must be generated for the clock and reset inputs. To generate these signals, logic should be utilized to minimize skew and maintain the rise and fall times as short as possible. This section describes a typical circuit that synthesizes the clock signal. RESET timing generation is discussed in the "RESET AND INITIALIZATION" section of this chapter.

Clock Generation

Figure 3-9 shows an example of a clock generator that produces two clock pulses, one double the frequency of the other with the skew between the pulses in the range of 1 to 3 ns. This particular circuit produces a 32-MHz clock at a 50% duty cycle. The circuit design consists of four devices: an oscillator, a pulse shaping network, a synchronous up/down counter, and a NAND gate driver. The output of the 64-MHz hybrid clock oscillator connects to the pulse shaping network (two NAND gates in series), which in turn feeds into the clock input of the up/down counter. This counter produces a 32-MHz CLK2 output signal and a 16-MHz CLK output signal. Because the outputs of the counter are synchronous, the skew between CLK2 and CLK is typically less than 2 ns. To provide adequate signal margin and maintain fast rise and fall times, the two clock signals are conditioned by the NAND gate driver. The timing waveforms of the clock circuit are shown in Figure 3-10.

If the opposite phase CLK is preferred, the U/\bar{D} pin can be connected to V_{cc} .

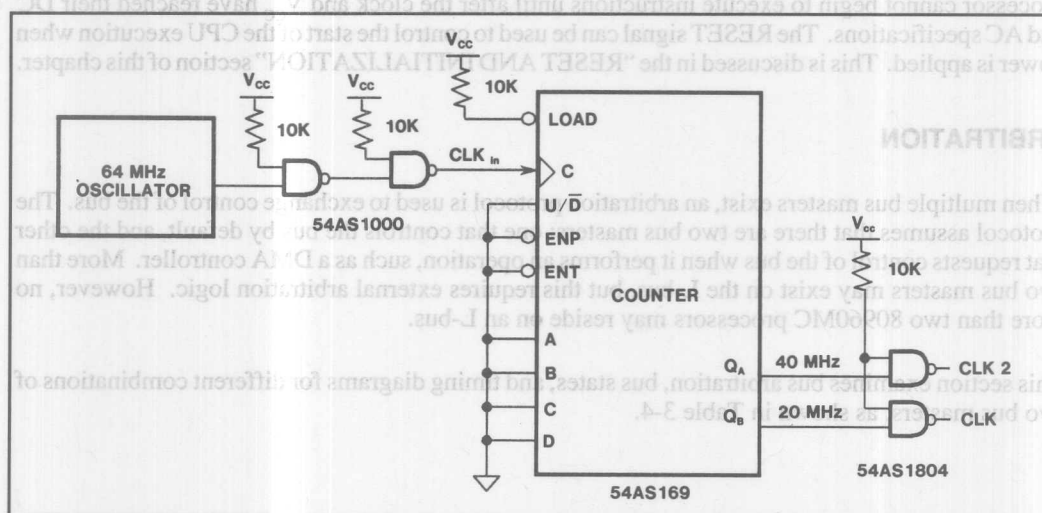


Figure 3-9: Clock Generation Circuit

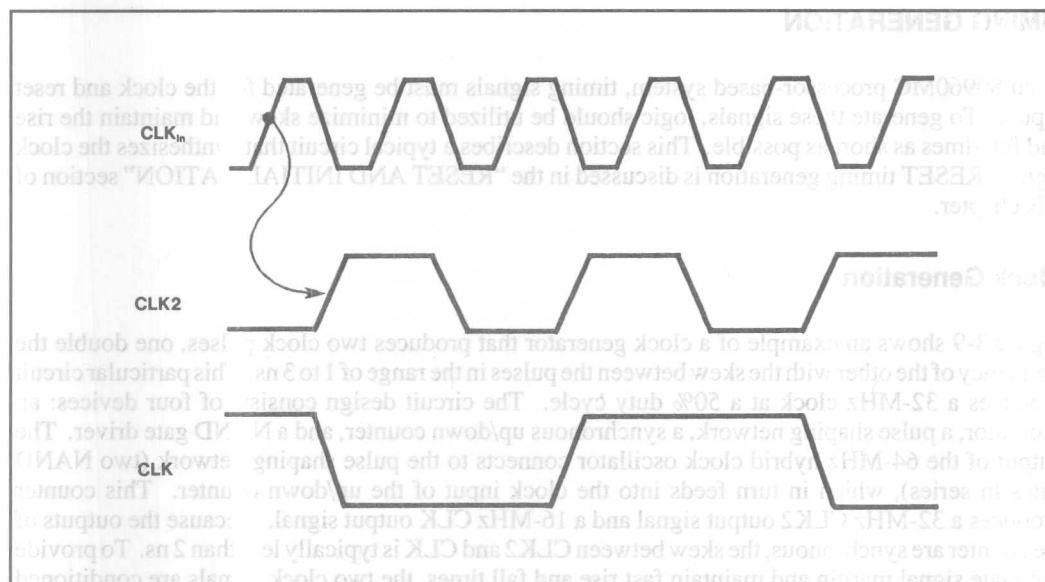


Figure 3-10: Clock Timing Waveforms

The hybrid clock oscillator typically requires 50 ms to stabilize after power is applied. The 80960MC processor cannot begin to execute instructions until after the clock and V_{cc} have reached their DC and AC specifications. The RESET signal can be used to control the start of the CPU execution when power is applied. This is discussed in the “RESET AND INITIALIZATION” section of this chapter.

ARBITRATION

When multiple bus masters exist, an arbitration protocol is used to exchange control of the bus. The protocol assumes that there are two bus masters: one that controls the bus by default, and the other that requests control of the bus when it performs an operation, such as a DMA controller. More than two bus masters may exist on the L-bus, but this requires external arbitration logic. However, no more than two 80960MC processors may reside on an L-bus.

This section examines bus arbitration, bus states, and timing diagrams for different combinations of two bus masters, as shown in Table 3-4.

Single 80960MC Processor On The L-Bus

For the first case, the 80960MC processor controls the L-bus, and a master I/O peripheral, such as a DMA controller, requests control of the bus. The 80960MC processor and the I/O peripheral exchange control of the bus with two signals: the hold request (HOLD) and hold acknowledge (HLDA) signals.

HOLD is an input signal of the 80960MC processor, which indicates that the master I/O peripheral is requesting control of the L-bus. When HOLD is asserted, the 80960MC processor surrenders control of the bus after it completes the current bus transaction. The 80960MC processor acknowledges transfer of control of the L-bus to the requesting bus master by asserting HLDA.

State Diagram

Figure 3-11 shows the state diagram for an L-bus with two bus masters: an 80960MC processor, and an I/O peripheral device. This state diagram includes a hold state (T_h) in addition to the five basic states described in the "BASIC L-BUS STATES" section of this chapter. The 80960MC processor enters the T_h state when it surrenders control of the bus. It can enter the T_h state from the T_i , T_r , T_d , or T_w state. When the 80960MC processor regains control of the L-bus, it enters the T_a state if a new request is pending or a T_i state if no new request is pending.

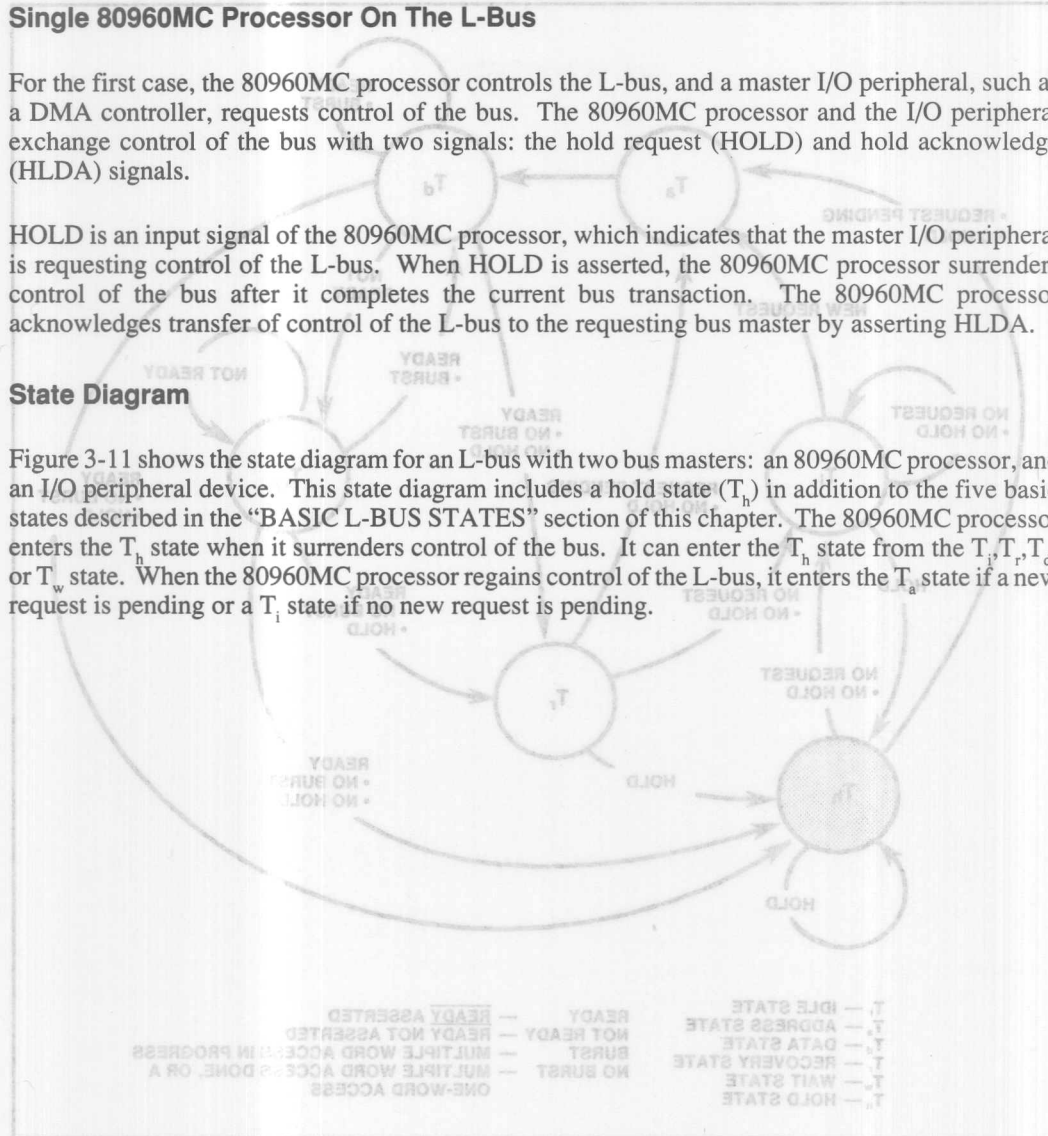


Figure 3-11: L-Bus States with Arbitration

Arbitration Timing

Figure 3-12 shows the arbitration timing diagram. The initial "T" state represents a T_i , T_a , T_d , T_r , or T_w state of a previous transaction as specified in the L-bus state diagram shown in Figure 3-11. The 80960MC processor receives a request to relinquish control of the bus when HOLD is asserted. After the 80960MC processor completes the current transaction, it responds to the request by floating the

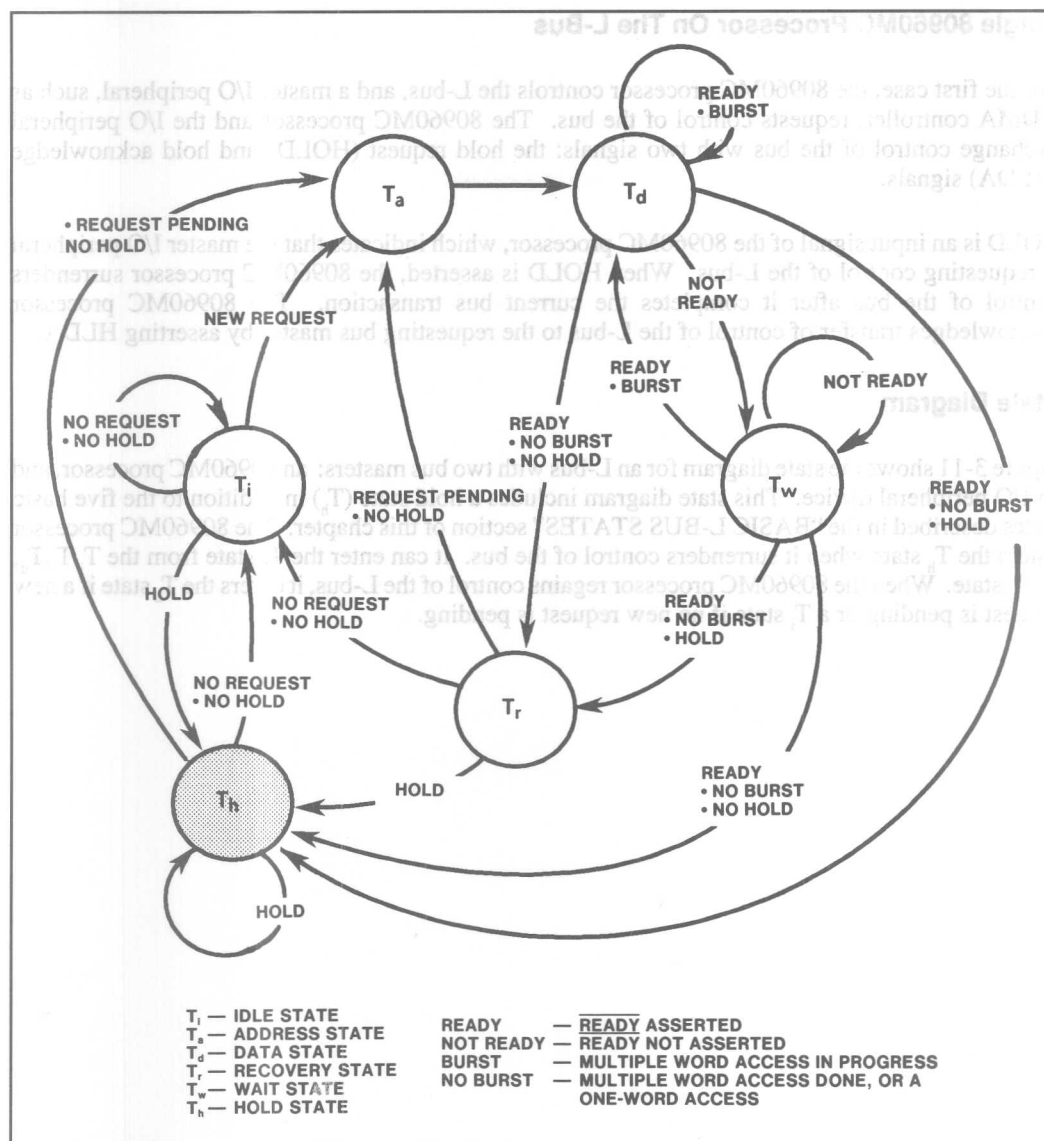


Figure 3-11: L-Bus States with Arbitration

Arbitration Timing

Figure 3-12 shows the arbitration timing diagram. The initial “T” state represents a T_d , T_w , T_r , or T_i state of a previous transaction as specified in the L-bus state diagram shown in Figure 3-11. The 80960MC processor receives a request to relinquish control of the bus when HOLD is asserted. After the 80960MC processor completes the current transaction, it responds to this request by floating the

three-state output signals and deasserting the open drain output signals. The HLDA output signal, however, remains active and is asserted as the 80960MC processor enters a T_h state. During the T_h state, the CPU ignores all input signals except HOLD and RESET. When the HOLD input signal is deasserted, the 80960MC processor exits the T_h state, deasserts HLDA, and enters a T_a state if a request is pending or a T_i state if no request is pending.

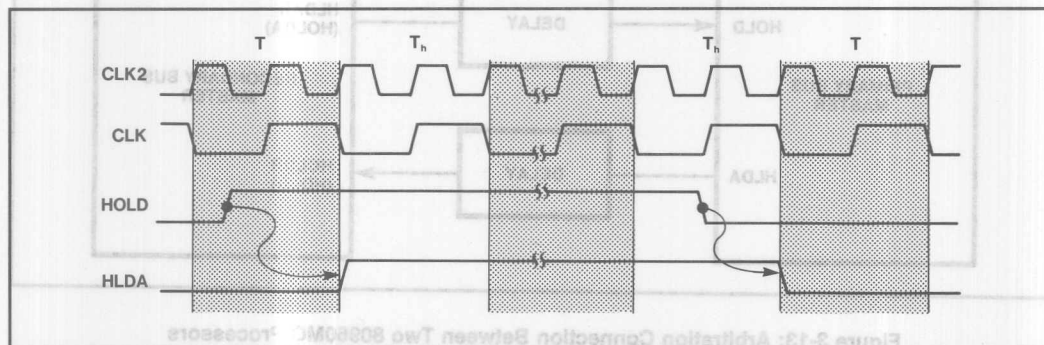


Figure 3-12: Arbitration Timing Diagram For A Bus Master

Two 80960MC Processors On The L-Bus

For the next case, two 80960MC processors reside on the L-bus. During initialization, Local Processor Number One (LPN1) is designated as the Primary Bus Master (PBM), and Local Processor Number Zero (LPN0) is designated as the Secondary Bus Master (SBM). The exchange protocol that is used guarantees that neither device is kept off the bus indefinitely.

The 80960MC processors use two pins for bus arbitration: the HOLD input pin, and the HLDA output pin. However, these input and output pins are interpreted differently for the Secondary Bus Master. When the SBM is initialized, the pin normally used for the HOLD input signal is interpreted as the Hold Acknowledge Request (HLDAR) input signal. The assertion of HLDAR indicates that the PBM relinquished control of the L-bus. Similarly, the HLDA output signal of the SBM is interpreted as the hold request (HOLDR) output signal. The SBM asserts HOLDR to request acquisition of the L-bus. Thus, bus arbitration between two 80960MC processors can be accomplished by connecting HOLD of the PBM to HOLDR of the SBM, and HLDA of the PBM to the HLDAR of the SBM, as shown in Figure 3-13.

When using the connection shown in Figure 3-13, a delay must be inserted between the input and output signals because the minimum output float delay, (shown as T_9 in the 80960MC Data Sheet), is less than the minimum hold time of the input signals, (shown as T_{11} in the 80960MC Data Sheet). The delay time necessary to meet the specified input setup and hold times can be calculated by using the following equation:

$$T_{11}(\text{min}) - T_9(\text{min}) < \text{Delay} < 2 \cdot T_{11}(\text{min}) - T_{12}(\text{min}) - T_6(\text{max})$$

The “T” numbers used in the above equation refer to timing parameters listed in the 80960MC Data Sheet. Consult the data sheet for actual timing values.

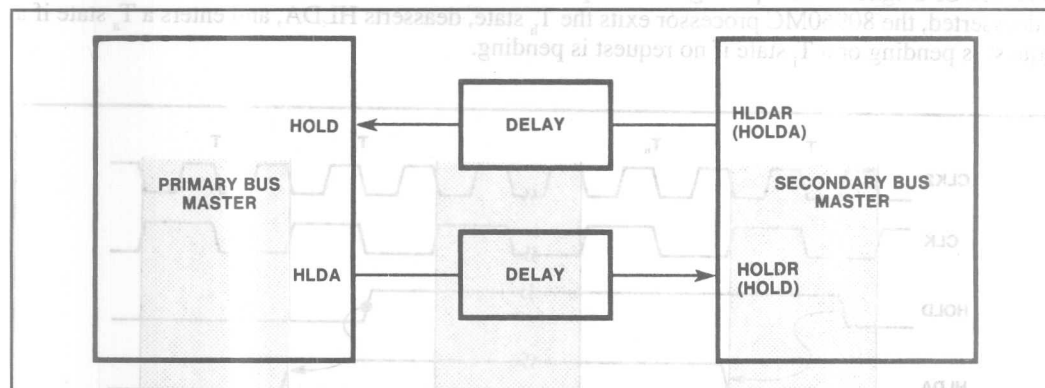


Figure 3-13: Arbitration Connection Between Two 80960MC Processors

Bus States For Two 80960MC Processors

The state diagram for the SBM is shown in Figure 3-14. Because there are two 80960MC processors, the $\overline{\text{LOCK}}$ signal is included in the state diagram. The SBM requests control of the L-bus by asserting HOLDR and subsequently enters the hold request (T_{hr}) state provided that the bus is not locked (locked means that the PBM is currently executing a Read-Modify-Write operation and has asserted the $\overline{\text{LOCK}}$ signal). The SBM remains in the T_{hr} state until it acquires control of the L-bus by receiving HLDAR . The SBM returns to the T_i state and deasserts HOLDR if the PBM asserts $\overline{\text{LOCK}}$ to execute a Read-Modify-Write operation.

The SBM gains control of the bus when HLDAR is asserted provided that the bus is not locked. After gaining control of the L-bus, the SBM performs the requested transaction and, if necessary, enters a T_w state. At the end of a transaction, the SBM goes to the T_r state and deasserts HOLDR for at least one processor clock cycle to allow another peripheral bus master to gain access if needed. If another request is pending, the SBM enters the T_{hr} state and asserts HOLDR provided the bus is not locked. If no request is pending the SBM returns to the T_i state. The PBM never forces the SBM off the bus.

Arbitration Timing for Two 80960MC Processors on the L-Bus

Figure 3-15 shows the timing diagram for acquiring and relinquishing the L-bus by a SBM. The SBM enters into the Hold Request (T_{hr}) state and asserts the HOLDR signal. It remains in the T_{hr} state until HLDAR is asserted, which indicates that the SBM has gained control of the L-bus. At the end of the transaction, the SBM enters the T_r state and deasserts HOLDR . Except for HOLDR , the output signals of the SBM go into a high impedance state or are deasserted for the case of open-drain outputs.

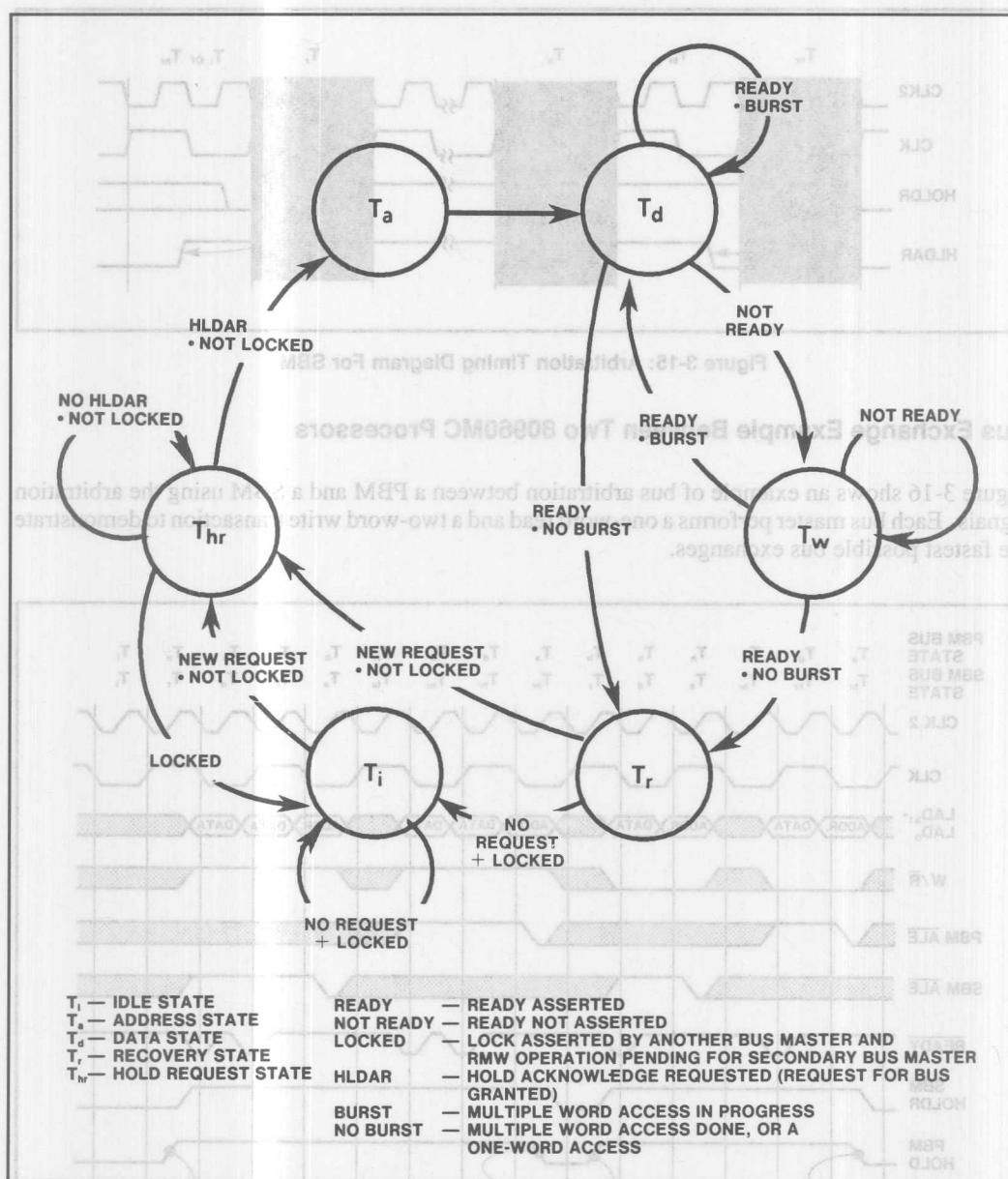


Figure 3-14: L-Bus States For Secondary Bus Master

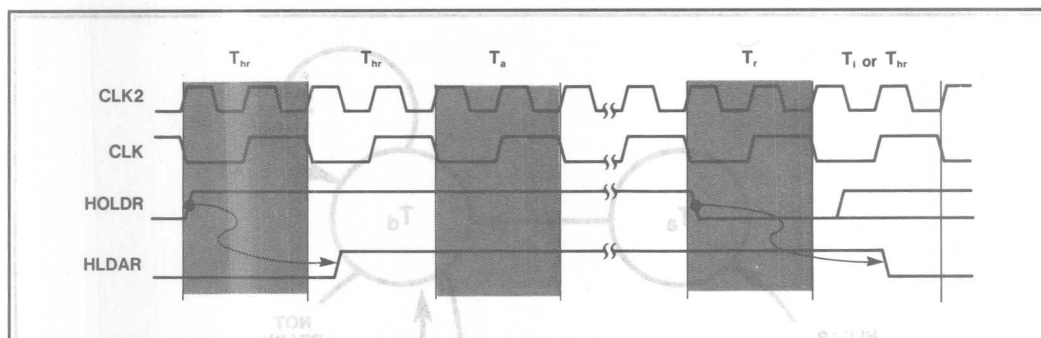


Figure 3-15: Arbitration Timing Diagram For SBM

Bus Exchange Example Between Two 80960MC Processors

Figure 3-16 shows an example of bus arbitration between a PBM and a SBM using the arbitration signals. Each bus master performs a one-word read and a two-word write transaction to demonstrate the fastest possible bus exchanges.

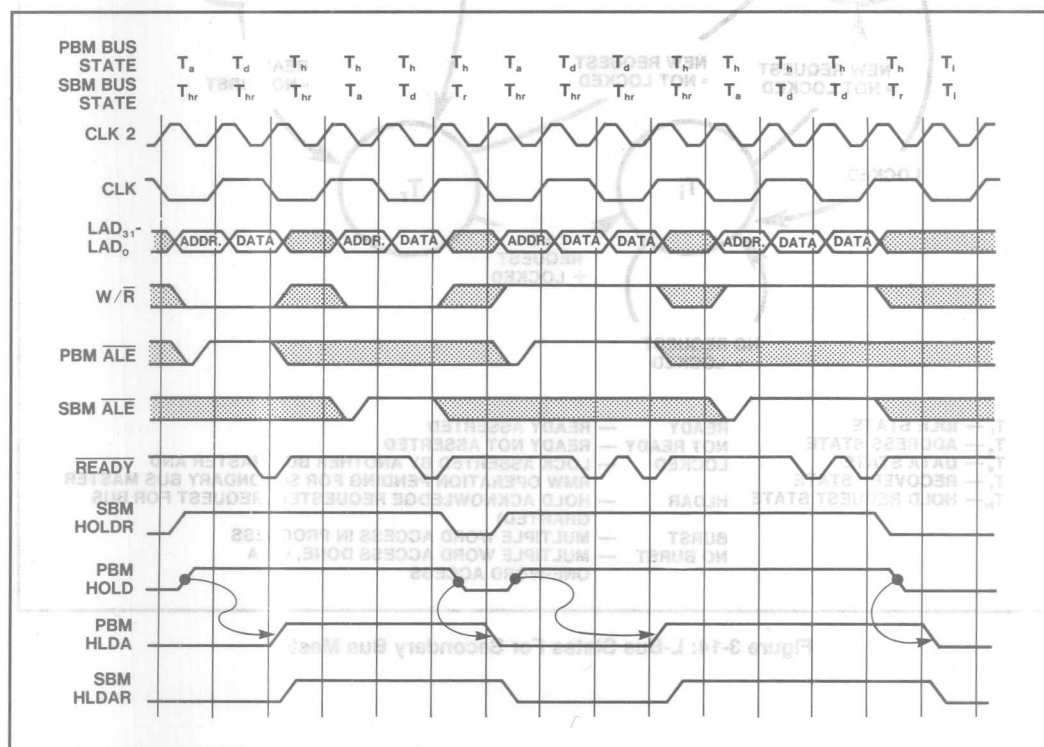


Figure 3-16: Example of a Bus Exchange Transaction

While the PBM is performing a read transaction, the SBM requests control of the L-bus by asserting **HOLDR** and entering the T_{hr} state. It remains in this state until the PBM grants the request by asserting **HLDA** after the read transaction is completed. After granting the request, the PBM enters the T_h state and remains in this state until its **HOLD** signal is deasserted. When the SBM completes the read transaction, it deasserts **HOLDR** and gives control back to the PBM.

The PBM now performs a two-word write transaction after deasserting the **HLDA**. The SBM requests control of the bus again by asserting the **HOLDR** signal and enters the T_{hr} state. When the PBM completes the two-word write transaction, it grants the request by asserting **HLDA** and enters the T_h state. The SBM receives the signal on the **HLDAR** input and performs a two-word write transaction. When the SBM completes the transaction, the control of the L-bus is transferred to the PBM, and both the PBM and the SBM enter the T_i state.

A Peripheral Device As The Default Bus Master

Another case exists where a peripheral device controls the L-bus, and the 80960MC processor requests control of the bus to perform operations. This alternative is not advisable because it hinders system performance. The exchange protocol is identical to the one described in the previous section. The 80960MC processor is a SBM and uses two pins for bus arbitration: the **HOLDR** input pin and the **HLDAR** output pin. The state diagram is similar to the one shown in Figure 3-14. The lock conditions are not used for this case, however.

The peripheral device grants control of the L-bus by asserting **HLDAR** when the SBM requests use of the L-bus. The peripheral device can obtain control of the L-bus again by deasserting **HLDAR**. If this occurs, the 80960MC processor surrenders control of the bus after it completes the current transaction, as shown in Figure 3-17. At that time, the 80960MC processor deasserts the **HOLDR** signal and places the other output signals into a high impedance state or a deasserted open drain level. The 80960MC processor may request access to the L-bus by asserting **HOLDR** again.

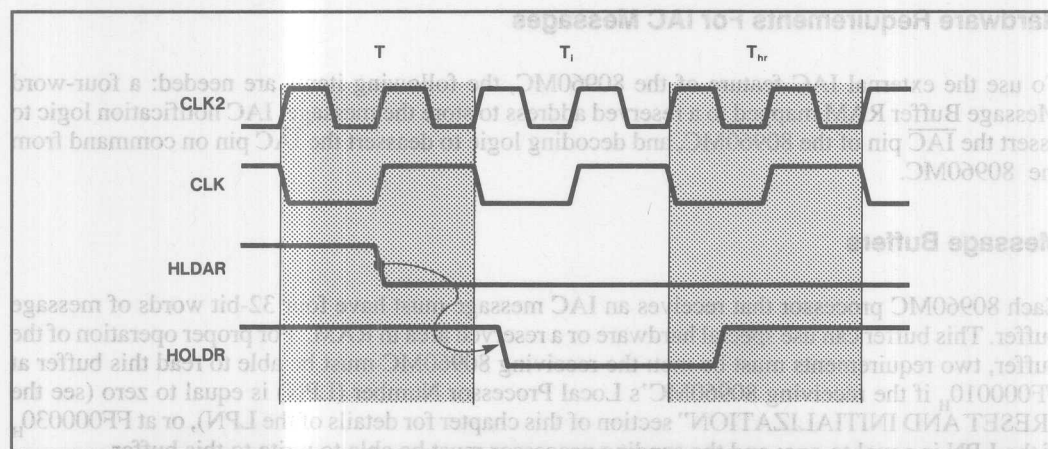


Figure 3-17: Forced Relinquishment Timing Diagram For A SBM

INTER-AGENT COMMUNICATION (IAC)

The IAC mechanism gives 80960MC processors the capability to send and receive messages to one another and to other bus agents. The IAC mechanism is essentially a **NON-MASKABLE** interrupt with pre-defined service routines. These routines are implemented in the 80960MC processor and are used to perform control functions such as purging the instruction cache, setting breakpoint registers, or stopping and starting the processor. By using IAC messages, external devices can remotely control the 80960MC. This allows easy integration of the 80960MC into system environments.

IAC messages can also be used to generate interrupts that behave exactly the same as hardwired interrupts. Since the interrupt vector is encoded in the IAC message, any of the 248 possible interrupt service routines can be invoked. For further information on IAC message definitions see the *80960MC Programmer's Reference Manual*.

Overview Of IAC Operations

Figure 3-18 shows a typical example of an IAC operation. In this case, an external processor gains control of the 80960MC by using an IAC operation. The external processor performs two functions: it writes the message in a buffer, called the message buffer; and it asserts the $\overline{\text{IAC}}$ pin of the 80960MC processor. Upon receipt of the $\overline{\text{IAC}}$ signal, the 80960MC processor stops executing its current process and performs a four-word burst read of the message buffer. After completing the read operation, the 80960MC processor automatically performs a one-word write operation to a pre-defined address to acknowledge the receipt of the message. The 80960MC processor then proceeds to perform the required action. When an IAC sender and recipient are the same physical processor (i.e. an 80960MC sending an IAC to itself), no L-bus activity is required. This allows software to move data or address's over the L-bus during the same bus cycle that it is executing the IAC, therefore increasing the effective throughput of the system.

Hardware Requirements For IAC Messages

To use the external IAC feature of the 80960MC, the following items are needed: a four-word Message Buffer RAM mapped to a reserved address to store the message, IAC notification logic to assert the $\overline{\text{IAC}}$ pin of the 80960MC, and decoding logic to deassert the $\overline{\text{IAC}}$ pin on command from the 80960MC.

Message Buffers

Each 80960MC processor that receives an IAC message must have four 32-bit words of message buffer. This buffer can use special hardware or a reserved area in RAM. For proper operation of the buffer, two requirements must be met: the receiving 80960MC must be able to read this buffer at FF000010_H if the receiving 80960MC's Local Processor Number (LPN) is equal to zero (see the "RESET AND INITIALIZATION" section of this chapter for details of the LPN), or at FF000030_H if the LPN is equal to one; and the sending processor must be able to write to this buffer.

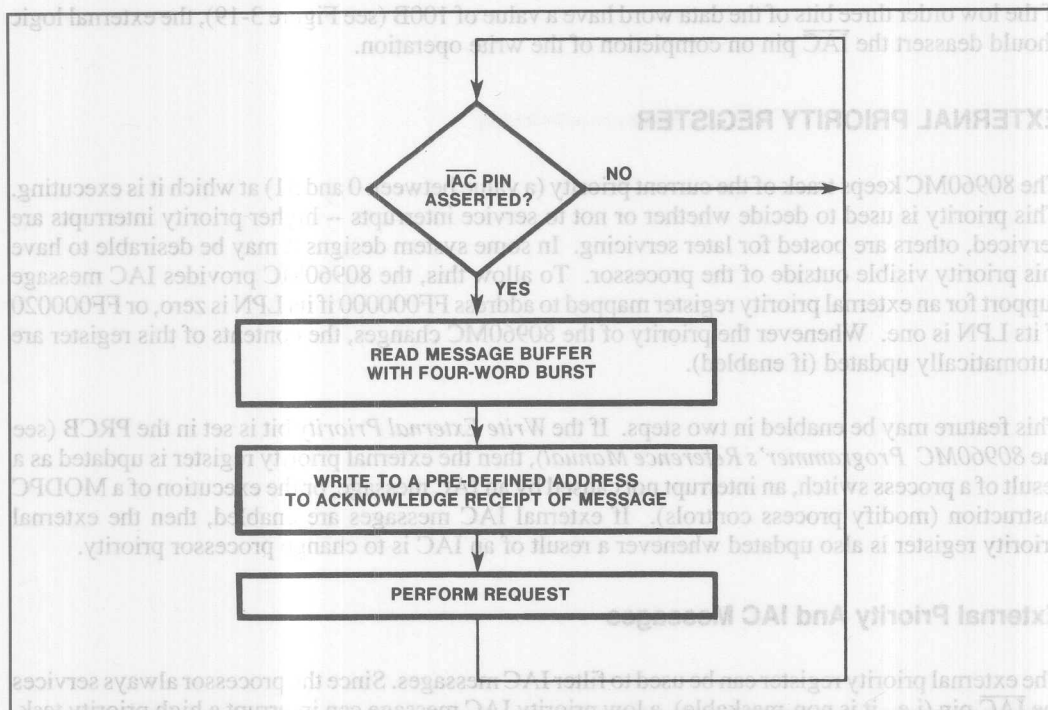


Figure 3-18: Example Flow Chart For An IAC Operation

IAC Pin Logic

When the IAC message buffer receives a message, IAC notification logic asserts the IAC pin and keeps it asserted. After the 80960MC processor reads the IAC message, it performs a one-word write to address FF000000H if its LPN is zero, or FF000020H if its LPN is one. This reserved address serves two functions: it causes external logic to deassert the IAC pin, and it maps to a register that contains the current processor priority. The 80960MC expects to write its priority into a 5-bit field of address FF000000 if its LPN is zero, or FF000020 if its LPN is one. To set the priority, the processor performs a one-word write operation in the form shown in Figure 3-19. The priority is contained in bit20-bit16, and bit3 is asserted to indicate that the priority is changed. It is necessary to use bit3 as a qualifier to distinguish priority write operations from IAC message acknowledgments, which use the same reserved address.

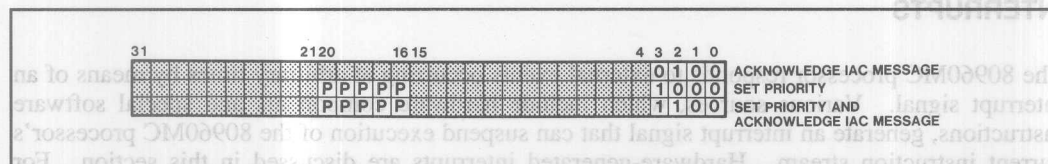


Figure 3-19: Data Settings

If the low order three bits of the data word have a value of 100B (see Figure 3-19), the external logic should deassert the $\overline{\text{IAC}}$ pin on completion of the write operation.

EXTERNAL PRIORITY REGISTER

The 80960MC keeps track of the current priority (a value between 0 and 31) at which it is executing. This priority is used to decide whether or not to service interrupts -- higher priority interrupts are serviced, others are posted for later servicing. In some system designs it may be desirable to have this priority visible outside of the processor. To allow this, the 80960MC provides IAC message support for an external priority register mapped to address FF000000 if its LPN is zero, or FF000020 if its LPN is one. Whenever the priority of the 80960MC changes, the contents of this register are automatically updated (if enabled).

This feature may be enabled in two steps. If the *Write External Priority* bit is set in the PRCB (see the *80960MC Programmer's Reference Manual*), then the external priority register is updated as a result of a process switch, an interrupt not caused by an IAC message, or the execution of a MODPC instruction (modify process controls). If external IAC messages are enabled, then the external priority register is also updated whenever a result of an IAC is to change processor priority.

External Priority And IAC Messages

The external priority register can be used to filter IAC messages. Since the processor always services the $\overline{\text{IAC}}$ pin (i.e., it is non-maskable), a low priority IAC message can interrupt a high priority task. To prevent this, a system can associate a priority with each IAC message. This priority can then be compared to the priority stored in the external priority register and used to decide whether or not to accept the IAC message. One way to associate a priority with an IAC message is to encode the message priority into the IAC message destination address as shown in Figure 3-20. The range of reserved addresses shown in Figure 3-20 have been set aside for this purpose.

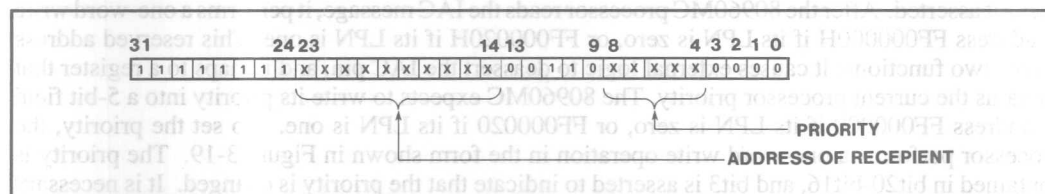


Figure 3-20: Physical Address Interpretation For IAC Messages

INTERRUPTS

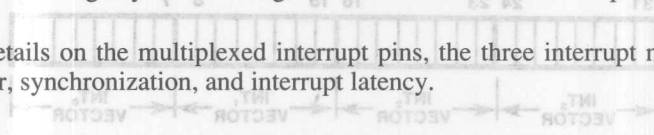
The 80960MC processor responds to external events occurring at arbitrary times by means of an interrupt signal. Various sources, which include hardware components and special software instructions, generate an interrupt signal that can suspend execution of the 80960MC processor's current instruction stream. Hardware-generated interrupts are discussed in this section. For complete information on software-generated interrupts, see the *80960MC Programmers Reference Manual*.

The 80960MC architecture provides a flexible interrupt structure. The processor can be interrupted using any of the three methods shown below:

- Receipt of a signal on any of the four direct interrupt input signal lines ($\overline{\text{INT}}_0$, INT_1 , INT_2 , and INT_3)
- Receipt of a signal on the interrupt request (INTR) line to obtain an external interrupt vector
- Receipt of an IAC message from a processor program or external source.

The choice of the method is determined by the setting in the on-chip Interrupt Control Register. Interrupt signals can occur during any bus state regardless of which method is implemented.

This section provides details on the multiplexed interrupt pins, the three interrupt methods, the Interrupt Control register, synchronization, and interrupt latency.



Interrupt Signals

The interrupt signals are multiplexed on four pins of the 80960MC processor: $\overline{\text{INT}}_0/\overline{\text{IAC}}$, INT_1 , INT_2/INTR , and INT_3/INTA . The on-chip Interrupt Control register determines how these pins are used (see "Interrupt Control Register" section of this chapter).

$\overline{\text{INT}}_0/\overline{\text{IAC}}$

This pin multiplexes the **Interrupt₀** and **Inter-agent Communication (IAC)** request input signals. The 80960MC processor interprets this input signal as either $\overline{\text{INT}}_0$ or $\overline{\text{IAC}}$. The $\overline{\text{INT}}_0$ signal indicates a request for interrupt service when it is asserted. The $\overline{\text{IAC}}$ signal denotes that an $\overline{\text{IAC}}$ message is waiting when it is asserted.

INT_1

The **Interrupt₁** input signal indicates a request for interrupt service when it is asserted.

INT_2/INTR

This pin multiplexes the **Interrupt₂** and **Interrupt Request** input signals. The 80960MC processor interprets this input signal as either INT_2 or INTR . The INT_2 signal indicates a request for interrupt service when it is asserted. The INTR signal indicates an interrupt request from an external interrupt controller. The 80960MC processor responds with an interrupt-acknowledge sequence. To ensure an interrupt, the INTR signal must remain asserted until the first cycle of the interrupt-acknowledge transaction.

INT_3/INTA

This pin multiplexes the **Interrupt₃** input signal and **Interrupt Acknowledge** output signal. The 80960MC processor uses this pin as the INT_3 input signal or as the INTA output signal. The Interrupt Control Register setting selects either the combination of INTR/INTA or $\text{INT}_2/\text{INT}_3$. The INT_3 input signal indicates a request for interrupt service when it is asserted. INTA acknowledges the interrupt request from an external interrupt controller. The INTA signal is latched by the 80960MC processor and remains valid during the T_d state and, if required, T_w states. This signal is an open drain output.

Interrupt Control Register

The 80960MC processor uses a 32-bit, on-chip Interrupt Control Register to define the function of the multiplexed interrupt pins. This 32-bit Interrupt Control Register allocates eight bits for each of the four direct interrupt signals ($\overline{\text{INT}}_0$, INT_1 , INT_2 , and $\overline{\text{INT}}_3$). The eight bits contain the vector number for each interrupt signal, as shown in Figure 3-21. The vector number is automatically read when one of the interrupt signals ($\overline{\text{INT}}_0$, INT_1 , INT_2 , and $\overline{\text{INT}}_3$) is activated. For example, when an interrupt is signaled on $\overline{\text{INT}}_0$, the 80960MC processor uses bit₇-bit₀ of the Interrupt Control register as the vector number.

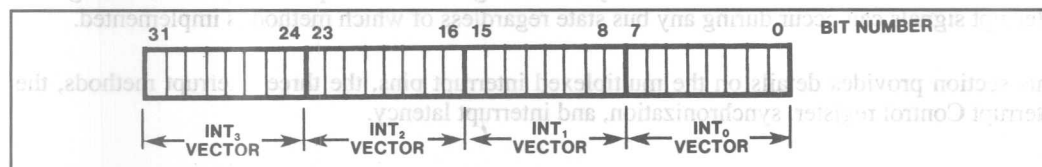


Figure 3-21: Interrupt Control Register

The 80960MC processor uses the data field corresponding to $\overline{\text{INT}}_0$ to determine identification of the $\overline{\text{INT}}_0/\text{IAC}$ input pin; a value of 00_{H} signifies the IAC function. If the data field corresponding to INT_2 has a value of 00_{H} , the 80960MC processor interprets the INT_2/INTR pin as the INTR input signal, and the $\overline{\text{INT}}_3/\text{INTA}$ pin as the INTA output signal. In other words, this setting specifies that the 80960MC processor should use these two pins for communication with an external interrupt controller. When used with an external interrupt controller, the data field corresponding to INT_3 should be set to FF_{H} . If the functions of INTR and $\overline{\text{INTA}}$ are selected, the direct interrupt pins $\overline{\text{INT}}_0$ and INT_1 can still be used.

The on-chip Interrupt Control register may be read and written by the Synchronous Load (synld) and Synchronous Move (synmov) instructions at the address $\text{FF000004}_{\text{H}}$ (see the *80960MC Programmer's Reference Manual*). The value of the data fields in the Interrupt Control register is $\text{FF000000}_{\text{H}}$ after initialization. This setting specifies that the four interrupt pins function as $\overline{\text{INTA}}$, INTR, INT_1 , and IAC.

Using The Four Direct Interrupt Pins

The 80960MC processor can be interrupted by asserting any of the four interrupt input signals ($\overline{\text{INT}}_0$, INT_1 , INT_2 , $\overline{\text{INT}}_3$). If the signals are simultaneously asserted, the 80960MC assumes that $\overline{\text{INT}}_0$ has the highest priority, followed by INT_1 , INT_2 , and $\overline{\text{INT}}_3$. Software should follow this convention when programming the Interrupt Control register. When the interrupt input signals are asserted, the 80960MC processor utilizes a vector number specified by the Interrupt Control register as an index to an entry in the interrupt table located in memory. For complete software information on this topic, see the *80960MC Programmer's Reference Manual*.

Using An External Interrupt Controller

The 80960MC processor can communicate with an external interrupt controller by performing an interrupt acknowledge sequence using the $\overline{\text{INTR}}$ and $\overline{\text{INTA}}$ signals. Figure 3-23 shows an example of the timing of an interrupt acknowledge sequence using the M8259A Programmable Interrupt Controller.

$\overline{\text{INTR}}$ is asserted by the M8259A and remains asserted until the 80960MC processor activates the $\overline{\text{INTA}}$ signal for the first time. When the 80960MC processor receives an interrupt request, the CPU completes the current transaction (or comes to some interruptible point), and asserts $\overline{\text{INTA}}$. $\overline{\text{INTA}}$ remains valid through the T_a , T_d , and T_w states. The first assertion of $\overline{\text{INTA}}$ triggers the M8259A to resolve priority among its interrupt requests.

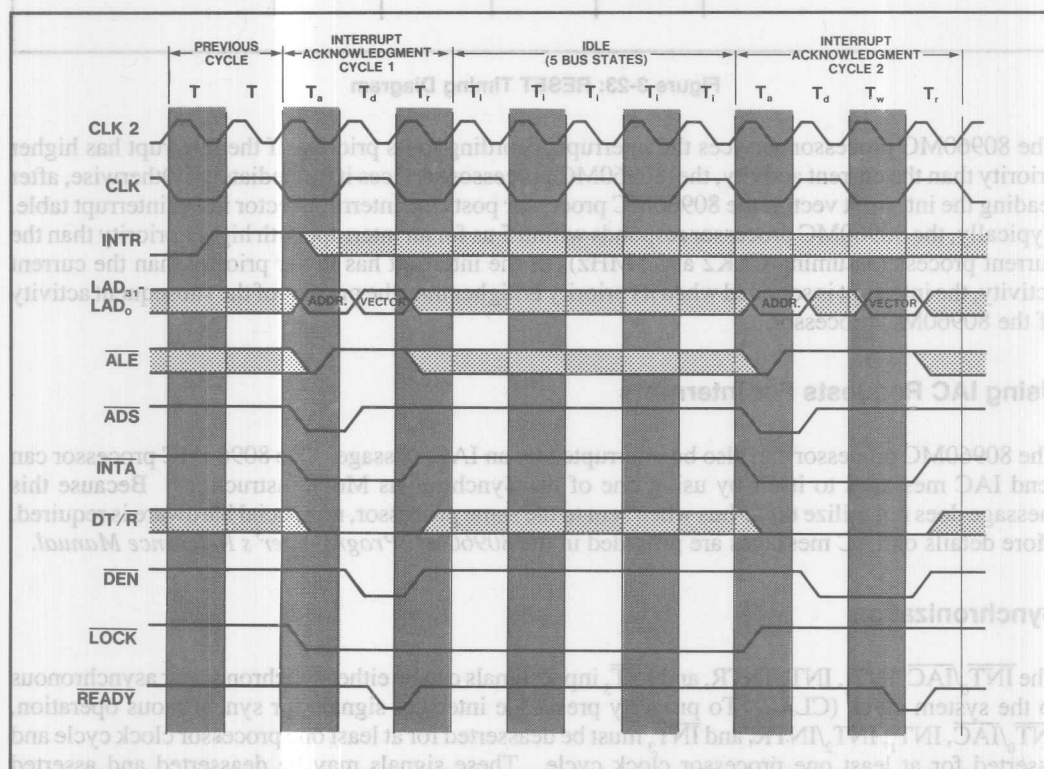


Figure 3-22: Timing Diagram For Interrupt Acknowledge Transaction

To compensate for the timing of the M8259A, the 80960MC processor inserts five T_i states before asserting the $\overline{\text{INTA}}$ again to read the interrupt vector. Figure 3-23 shows $\overline{\text{READY}}$ asserted without a wait state during the first Interrupt Acknowledgement cycle and with one wait state during the second Interrupt Acknowledgement cycle. In practice, the M8259A would require about four wait

states in both cycles. The address during the T_a state for both interrupt acknowledge cycles is $FFFFFFFC_H$. For more details, see the “M8259A Programmable Interrupt Controller” section in Chapter 5.

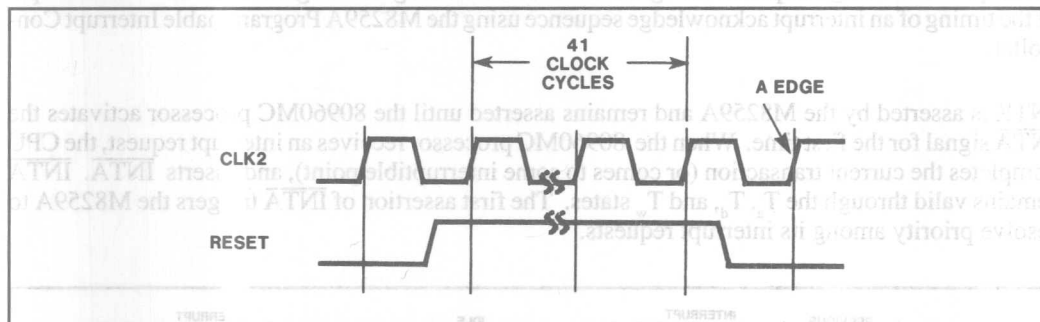


Figure 3-23: RESET Timing Diagram

The 80960MC processor services the interrupt according to its priority. If the interrupt has higher priority than the current activity, the 80960MC processor services it immediately. Otherwise, after reading the interrupt vector, the 80960MC processor posts the interrupt vector in the interrupt table. Typically, the 80960MC processor responds within 5 μs for an interrupt with higher priority than the current process (assuming CLK2 at 32 MHz). If the interrupt has lower priority than the current activity, the interrupt is serviced when its priority is higher than the priority of the subsequent activity of the 80960MC processor.

Using IAC Requests For Interrupts

The 80960MC processor can also be interrupted by an IAC message. The 80960MC processor can send IAC messages to itself by using one of the Synchronous Move instructions. Because this message does not utilize the L-bus when sent to the same processor, no special hardware is required. More details on IAC messages are provided in the *80960MC Programmer's Reference Manual*.

Synchronization

The $\overline{INT_0}/IAC$, INT_1 , $INT_2/INTR$, and $\overline{INT_3}$ input signals can be either synchronous or asynchronous to the system clock (CLK2). To properly preset the interrupt signals for synchronous operation, $\overline{INT_0}/IAC$, INT_1 , $INT_2/INTR$, and $\overline{INT_3}$ must be deasserted for at least one processor clock cycle and asserted for at least one processor clock cycle. These signals may be deasserted and asserted individually.

If the interrupt signals are asynchronous to CLK2, the 80960MC processor internally synchronizes them. For the CPU to recognize the asynchronous interrupt input signals, they must be preset by deasserting them for at least two processor clock cycles, and then asserting them for at least two processor clock cycles. These signals may be deasserted and asserted individually.

RESET AND INITIALIZATION

The system $\overline{\text{RESET}}$ signal provides an orderly way to start or restart the 80960MC processor. When the 80960MC processor detects the low-to-high transition of $\overline{\text{RESET}}$, it terminates all external activities and places the output pins in the high impedance state or deasserted condition. When the $\overline{\text{RESET}}$ signal falls low again, the 80960MC processor begins the initialization process and later starts fetching instructions from a specific address.

RESET Timing Requirements

To properly reset the 80960MC processor to a known state, the low-to-high transition of $\overline{\text{RESET}}$ must be asserted relative to any rising edge of CLK2 and remain asserted for at least 41 CLK2 cycles, as shown in Figure 3-24. $\overline{\text{RESET}}$ must be deasserted after the rising edge of CLK2, but prior to the next rising edge of CLK2. This establishes the next rising edge of CLK2 as edge A.

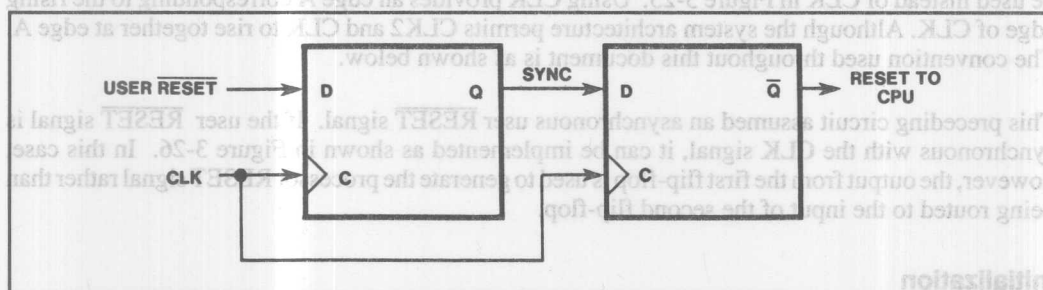


Figure 3-24: Asynchronous RESET Circuit

RESET Timing Generation

The $\overline{\text{RESET}}$ input signal to the 80960MC processor can easily be generated by implementing a synchronization circuit comprised of two D-type flip-flops, as shown in Figure 3-25.

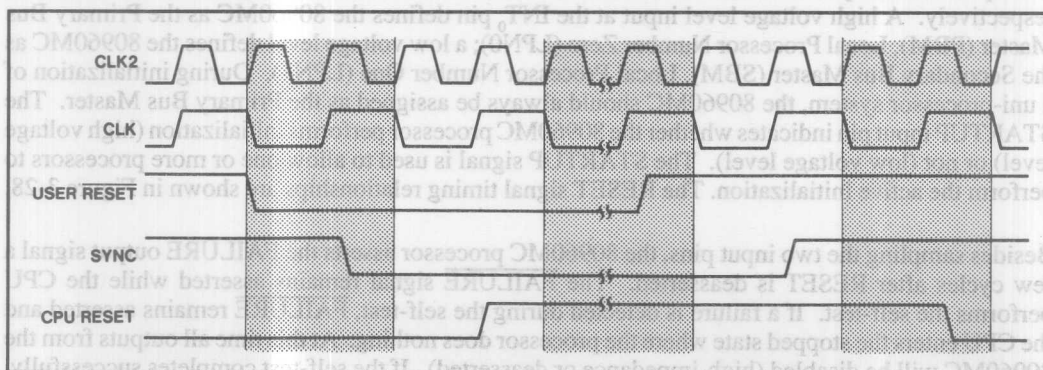


Figure 3-25: Diagram for RESET Timing Generation

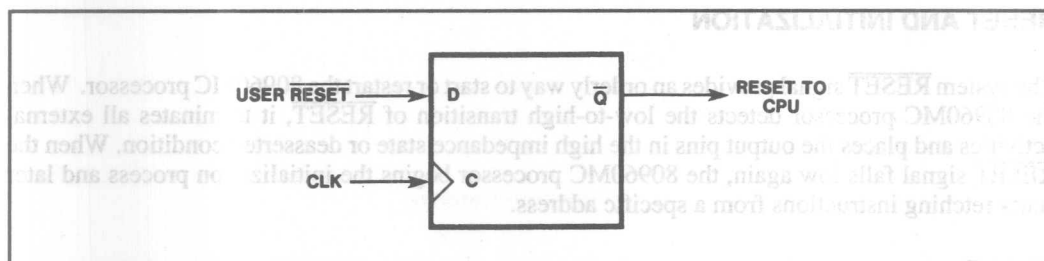


Figure 3-26: Synchronous RESET Circuit

The user RESET signal is synchronized with the CLK signal by applying CLK to the clock input of both flip-flops. To protect against a metastable RESET signal, the output of the first flip-flop, SYNC, is applied to the input of the second flip-flop. The output of the second flip-flop results in a processor RESET signal. The timing diagram for these signals is shown in Figure 3-25. CLK or CLK2 can be used instead of CLK in Figure 3-25. Using CLK provides an edge A corresponding to the rising edge of CLK. Although the system architecture permits CLK2 and CLK to rise together at edge A. The convention used throughout this document is as shown below.

This preceding circuit assumed an asynchronous user RESET signal. If the user RESET signal is synchronous with the CLK signal, it can be implemented as shown in Figure 3-26. In this case, however, the output from the first flip-flop is used to generate the processor RESET signal rather than being routed to the input of the second flip-flop.

Initialization

The initialization sequence of events is shown in Figure 3-27. When RESET is deasserted after a minimum of 41 CLK2 cycles, several actions take place: two input pins are sampled, the FAILURE output signal (see next section for the pin description) is asserted, and the self-test is performed.

When RESET is deasserted, the 80960MC processor samples the signals residing on the INT₀/IAC and the BADAC input pins (see the next section for the pin description of BADAC). At this time, these pins are interpreted as the Local Processor Number (LPN) and Startup (STARTUP) signals, respectively. A high voltage level input at the INT₀ pin defines the 80960MC as the Primary Bus Master (PBM), Local Processor Number Zero (LPN0); a low voltage level defines the 80960MC as the Secondary Bus Master (SBM), Local Processor Number One (LPN1). During initialization of a uni-processor system, the 80960MC should always be assigned as the Primary Bus Master. The STARTUP input pin indicates whether the 80960MC processor performs initialization (high voltage level) or not (low voltage level). The STARTUP signal is used to allow one or more processors to perform the active initialization. The RESET signal timing relationships are shown in Figure 3-28.

Besides sampling the two input pins, the 80960MC processor asserts the FAILURE output signal a few cycles after RESET is deasserted. The FAILURE signal remains asserted while the CPU performs the self-test. If a failure is detected during the self-test, FAILURE remains asserted and the CPU enters the stopped state where the processor does nothing. At this time all outputs from the 80960MC will be disabled (high-impedance or deasserted). If the self-test completes successfully, the CPU deasserts the FAILURE signal.

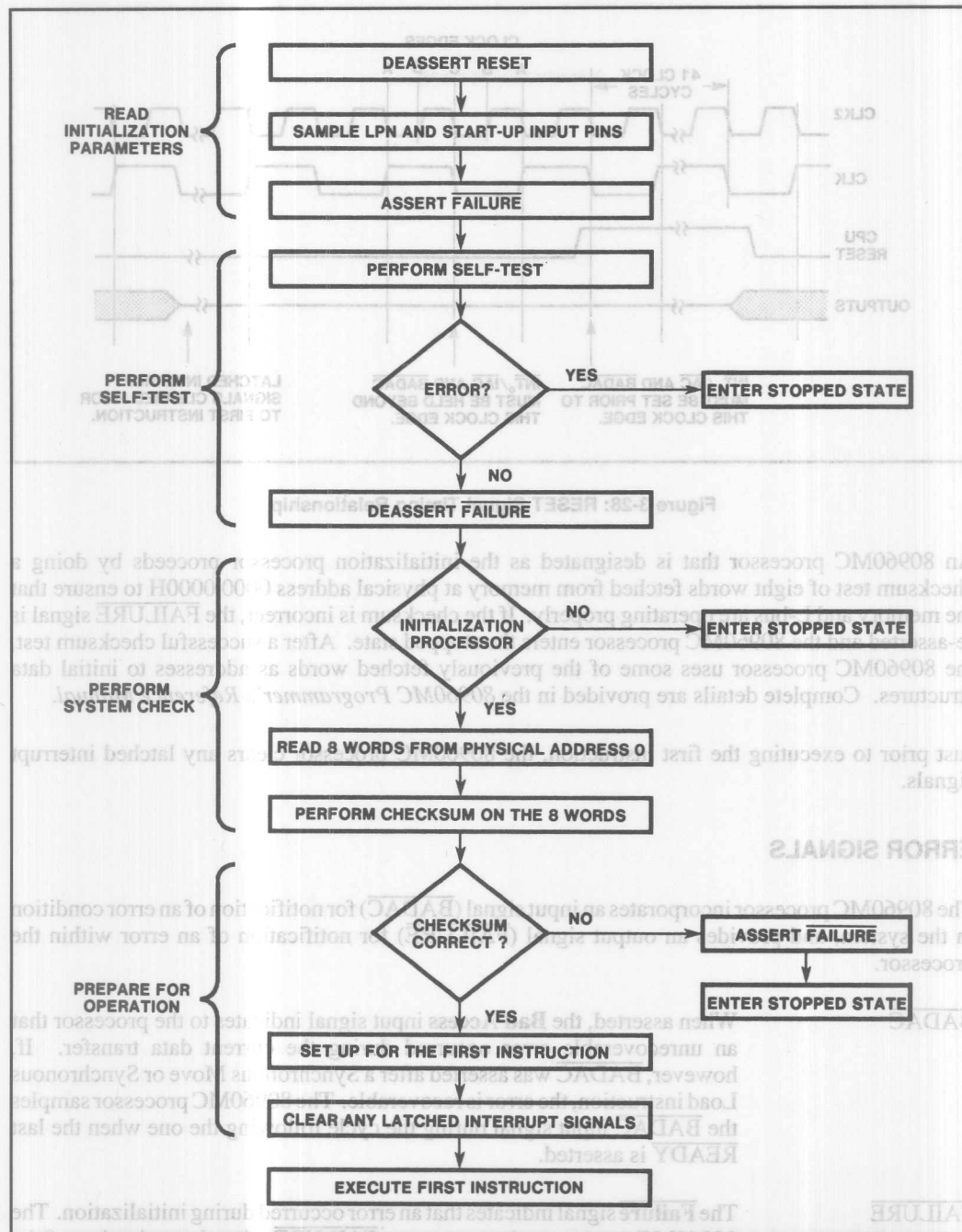


Figure 3-27: Initialization Flow Chart

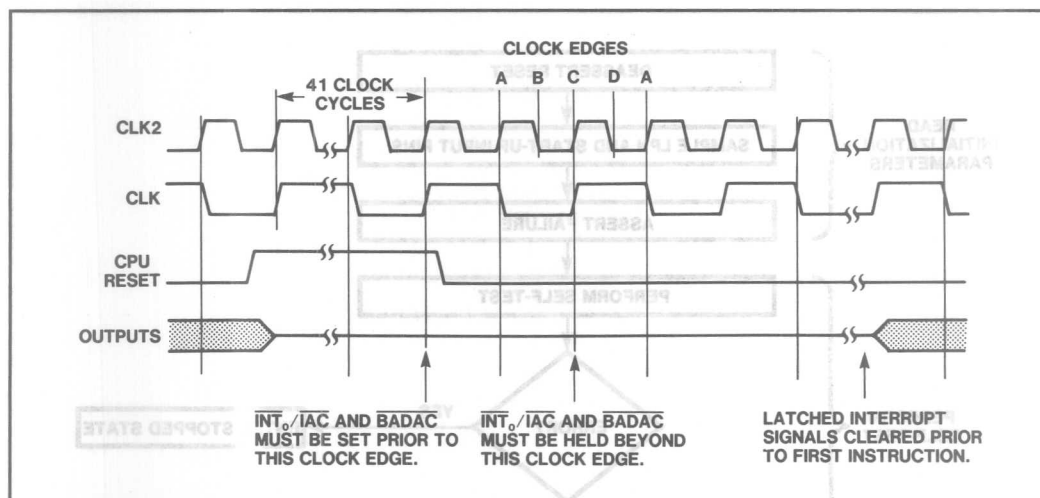


Figure 3-28: RESET Signal Timing Relationship

An 80960MC processor that is designated as the initialization processor proceeds by doing a checksum test of eight words fetched from memory at physical address 0000 0000H to ensure that the memory and L-bus are operating properly. If the checksum is incorrect, the **FAILURE** signal is re-asserted and the 80960MC processor enters the stopped state. After a successful checksum test, the 80960MC processor uses some of the previously fetched words as addresses to initial data structures. Complete details are provided in the *80960MC Programmer's Reference Manual*.

Just prior to executing the first instruction, the 80960MC processor clears any latched interrupt signals.

ERROR SIGNALS

The 80960MC processor incorporates an input signal (**BADAC**) for notification of an error condition in the system, and provides an output signal (**FAILURE**) for notification of an error within the processor.

BADAC

When asserted, the **Bad Access** input signal indicates to the processor that an unrecoverable error occurred during the current data transfer. If, however, **BADAC** was asserted after a Synchronous Move or Synchronous Load instruction, the error is recoverable. The 80960MC processor samples the **BADAC** input signal during the cycle following the one when the last **READY** is asserted.

FAILURE

The **Failure** signal indicates that an error occurred during initialization. The 80960MC processor always asserts **FAILURE** after the activation of the **RESET** signal. If a failure is detected during a self-test, **FAILURE** remains

asserted. Otherwise, the processor deasserts $\overline{\text{FAILURE}}$ after a successful self-test is performed. If the initial memory checksum is incorrect, the initialization process re-asserts $\overline{\text{FAILURE}}$ a second time, and keeps it asserted. $\overline{\text{FAILURE}}$ is an open drain output signal.

SUMMARY

The L-bus is a high speed 32-bit multiplexed bus with burst-transfer capability and is designed to operate with the high performance 80960MC processor. The L-bus consists of two signal groups: address/data, and control. These signal groups are utilized by the 80960MC processor to perform read, write, and burst transactions.

The arbitration, interrupt, and reset operations are related to the L-bus transactions. The arbitration operation transfers control of the L-bus to another bus master. Three methods are available to handle interrupts: by invoking the on-chip interrupt controller, by employing an external interrupt controller using the $\text{INTR}/\overline{\text{INTA}}$ signals, or by using an IAC message. The reset function sets the 80960MC processor to a known internal state after it successfully completes the self-test. These operations offer power and flexibility to hardware system design using the 80960MC processor.

This chapter focused on the L-bus and its relationship with the 80960MC processor. The next two chapters develop guidelines on interfacing memory and peripheral devices into the L-bus hardware system.

CHAPTER 4 MEMORY INTERFACE

The high-speed L-bus architecture has many features that enhance high-performance designs. In particular, the burst-transfer feature allows up to four successive 32-bit data word transfers at a maximum rate of one word every processor clock cycle. This chapter outlines approaches for memory designs that use these features, describes memory design considerations, analyzes the timing, and lists a number of useful examples. The concepts illustrated by these examples apply to a wide variety of memory system implementations.

BASIC MEMORY INTERFACE

Figure 4-1 shows the major logic blocks of the memory interface circuit. The data transceivers buffer the data to compensate for any slow devices that may be connected to the 80960MC processor. The address latches demultiplex the address/data signals from the 80960MC processor and latch the address. The address decoder selects the appropriate memory device from the latched address. To accommodate a memory burst transaction, the burst logic decrements the word count, increments the local address lines 3 and 2 (LAD_3 and LAD_2), and generates a CYCLE-IN-PROGRESS signal. The timing control generates a \overline{READY} signal and other specific signals required by a particular memory device. The byte enable latch stores the byte enable signals.

Although not part of the basic memory interface, the DRAM controller, SRAM interface, DRAM, SRAM, and EPROM are included in Figure 4-1 for completeness. In a hardware system the DRAM, SRAM, and EPROM are typically located in separate subsystems.

Although the memory interface circuit can be designed using programmable logic, gate arrays, or other custom logic, the examples use standard components wherever possible to illustrate the design concepts.

Data Transceivers

Standard 8-bit transceivers can be used to provide isolation and additional drive capability for the L-bus. Transceivers can be used to prevent bus contention that can occur if some memories are slow to remove data from the L-bus after a read operation. For example, if a write operation follows a read operation, the 80960MC processor may drive the L-bus before a slow device has removed its output data, potentially causing a current spike on the power and ground lines. Transceivers, however, can be omitted if the data float time of the device is short enough and the load does not exceed the 80960MC device specifications.

The data transceivers can be controlled by two signals from the 80960MC processor: data transmit/receive (DT/R) and data enable (DEN). DT/R indicates the direction of data flow and DEN enables the transceivers.

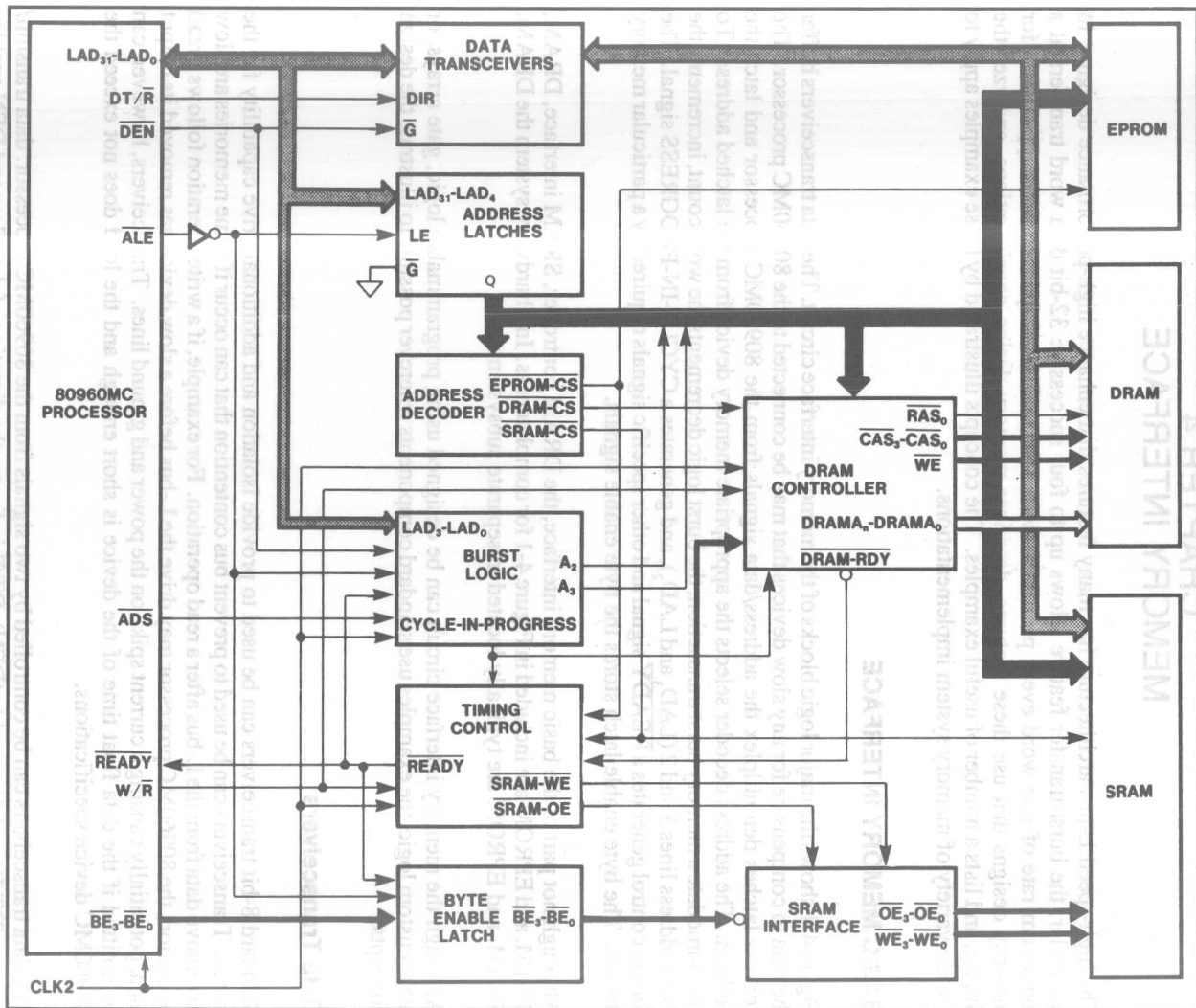


Figure 4-1: Simplified Block Diagram for Memory Interface Logic

Address Latch/Demultiplexer

Conventional transparent latches can be used to demultiplex the address/data lines of the 80960MC processor and to hold the address constant during the memory operation. The latch is controlled by the $\overline{\text{ALE}}$ signal from the 80960MC processor. $\overline{\text{ALE}}$ passes through an inverter, so that when $\overline{\text{ALE}}$ goes low, the address flows through the latch. The low-to-high transition of $\overline{\text{ALE}}$ can be used to latch the address. The output enable of the latch can be tied to ground. The lower four address lines (LAD_3 - LAD_0) are latched by the burst logic.

Address Decoder

The 80960MC processor accesses both memory and I/O devices by supplying a 32-bit address and a read/write command. The address decoder determines which particular memory or I/O device is selected by decoding the address lines. The following discussion focuses on memory selection, and the "Address Decoder" section in Chapter 5 discusses I/O device selection using memory-mapped I/O techniques.

The memory address can be divided into regions where one region can apply to EPROM or ROM, another to RAM, and another to the I/O registers. In a 80960MC-based system the ROM address space is likely to start at address 0000 0000H because the CPU begins execution at this address. The RAM or I/O regions can start at any other address in the 4G-byte address range except for addresses FFFFFFFFH through FFFFFFFFH, which the 80960MC processor reserves for inter-agent communication.

Because of the large address range of the 80960MC processor, the address can be divided into word address bits and chip select bits. Typically the higher-order address bits are decoded to generate the selection signal for ROM, RAM, or I/O devices.

The address decoder can be located either before or after the address latches. Usually, it is placed after the latches, so that the chip-select signal does not need to be latched. Figure 4-1 shows the block diagram of the address decoder placed behind the address latches.

Burst Logic

To enhance system performance, the 80960MC processor performs burst transactions that transfer up to four data words at a maximum rate of one word every clock cycle. A DRAM controller can be designed that takes advantage of the burst-transfer capability by using the static column mode or nibble mode features of the DRAM (see the "DRAM Controller" section of this chapter). This DRAM controller requires a signal, called CYCLE-IN-PROGRESS, to identify the start and end of a memory cycle. The burst logic generates the CYCLE-IN-PROGRESS signal.

Figure 4-2 shows the flow chart for the burst logic. If $\overline{\text{ADS}}$ is low and $\overline{\text{DEN}}$ is high, then the burst logic latches LAD_3 through LAD_0 , and asserts the CYCLE-IN-PROGRESS signal. The burst logic checks the SIZE signals (LAD_1 and LAD_0). If the value of the SIZE signals equal zero, then the burst logic runs one memory cycle, and terminates the CYCLE-IN-PROGRESS signal. If the value of the

SIZE signals do not equal zero, the burst logic runs one memory cycle, increments the lower two latched address's (A_2 and A_3), and decrements the SIZE value. When this is finished, the burst logic checks the value of the SIZE signals again.

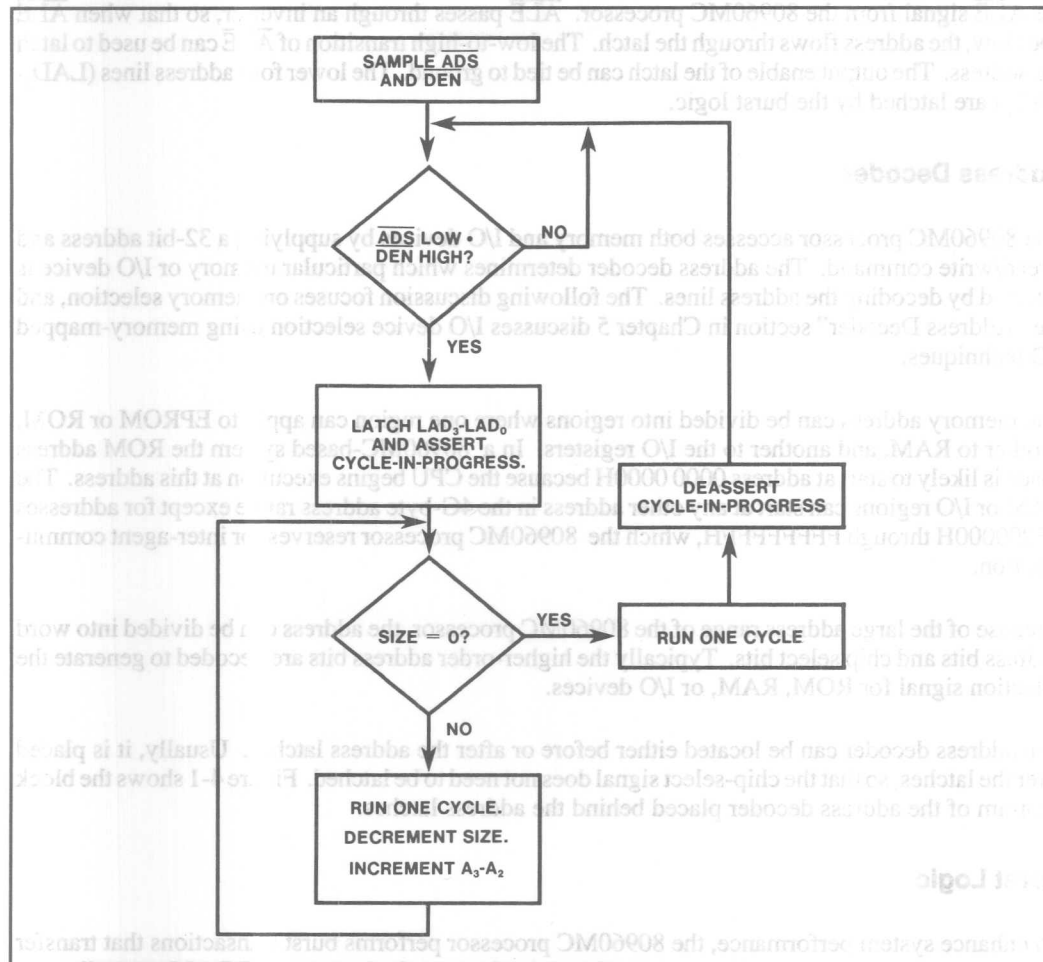


Figure 4-2 Burst Logic Flow Chart

The burst logic can be used with EPROM, SRAM, DRAM memories. However, it cannot be used in the DRAM static column or nibble modes, because they do not support burst transactions. Because the 80960MC processor ensures that a burst transaction cannot exceed four words or cross a 16-byte boundary, incrementing LAD_3 and LAD_2 after a single data word transfer makes the burst transfer transparent to the memory devices.

Timing Control Logic

The timing control logic accommodates memory devices that cannot transfer information at the maximum bus rate by inserting wait states until the data becomes available. The timing control logic consists of a counter and timing logic, as shown in Figure 4-3. The counter produces a 4-bit binary count. The count begins when the **CYCLE-IN-PROGRESS** signal is asserted. The timing logic asserts **READY** at the appropriate time based upon the count, the **EPROM-CS**, and the **SRAM-CS** signals. For a burst transfer, **READY** resets the counter to properly time a **READY** signal for the next data transfer. When **CYCLE-IN-PROGRESS** is deasserted, the clock counting is terminated.

Because the timing of DRAM is more complicated, the DRAM controller generates a **DRAM-RDY** signal to the timing control logic. In addition, the clock count, the **W/R** command, and **SRAM-CS** signal can also be used to generate **SRAM-WE** and **SRAM-OE** signals.

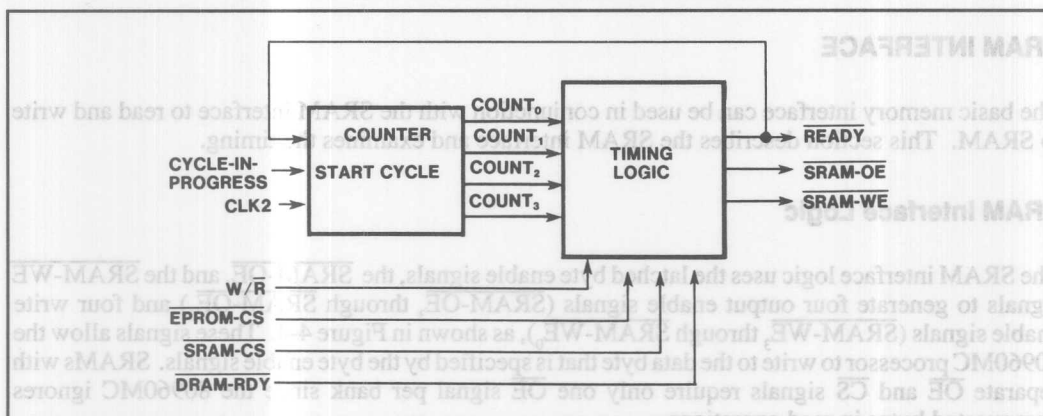


Figure 4-3: Memory Timing Control Block Diagram

Byte Enable Latch

The byte enable latch holds the byte enable signals constant until the DRAM controller or SRAM interface uses the signals. As mentioned in the "L-Bus Signal Groups" section in Chapter 3, the byte enable signals specify which bytes (up to four) on the 32-bit data bus are transferred during the data cycle. Each individual byte enable signal selects eight data lines as shown in Table 4-1.

The byte enable signals are valid from the 80960MC processor before data is transferred. These signals are asserted during the address cycle for the first data word transfer; they are asserted again during the first data cycle for the second word transfer; the second data cycle for the third word transfer; and the third data cycle for the fourth word transfer. For each word, the byte enable signals remain valid throughout every data or wait cycle until **READY** is asserted. After **READY** is asserted, the byte enable signals change during the next processor clock cycle.

The $\overline{\text{ALE}}$ signal can be used to latch the first byte enable signals. $\overline{\text{READY}}$ can be used to latch the other byte enable signals for each word.

Table 4-1: Byte Enable Signal Decoding

Byte Enable Signal	Address Line Selection
$\overline{\text{BE}}_0$	LAD ₇ -LAD ₀
$\overline{\text{BE}}_1$	LAD ₁₅ -LAD ₈
$\overline{\text{BE}}_2$	LAD ₂₃ -LAD ₁₆
$\overline{\text{BE}}_3$	LAD ₃₁ -LAD ₂₄

SRAM INTERFACE

The basic memory interface can be used in conjunction with the SRAM interface to read and write to SRAM. This section describes the SRAM interface and examines the timing.

SRAM Interface Logic

The SRAM interface logic uses the latched byte enable signals, the $\overline{\text{SRAM-OE}}$, and the $\overline{\text{SRAM-WE}}$ signals to generate four output enable signals ($\overline{\text{SRAM-OE}}_3$ through $\overline{\text{SRAM-OE}}_0$) and four write enable signals ($\overline{\text{SRAM-WE}}_3$ through $\overline{\text{SRAM-WE}}_0$), as shown in Figure 4-4. These signals allow the 80960MC processor to write to the data byte that is specified by the byte enable signals. SRAMs with separate $\overline{\text{OE}}$ and $\overline{\text{CS}}$ signals require only one $\overline{\text{OE}}$ signal per bank since the 80960MC ignores unrequested bytes in read operations.

SRAM Timing Considerations

This section analyzes the critical timing paths of the SRAM control signals. From the critical path, the timing equations can be derived to determine the memory access time for no wait state operation.

When evaluating critical timing paths, the timing calculations should use worst-case data sheet parametric specifications, rather than typical specifications. By using worst-case timing values, reliable operation is assured over all variations in temperature, voltage, and individual device characteristics. These timing values are determined by assuming the maximum propagation delay to latch an address, select a memory device, and pass through data buffers and transceivers.

Figure 4-5 shows the critical timing path for a one-word SRAM read operation. The diagram consists of three time periods: the address setup period (T_{addrset}), the memory response period (T_{mem}), and the data return period (T_{dataset}). Note that the timing for the read command and output control signals does not enter into the critical timing path.

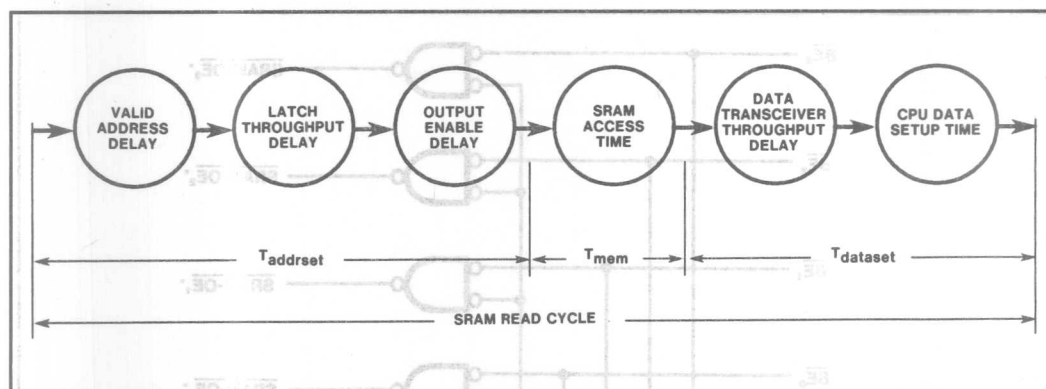


Figure 4-5: Critical Timing Path for SRAM Read Operation

For a no wait state operation, the data transfer word must be completed in two system clock (CLK) cycles. The minimum time period for a no wait state operation ($T_{\text{mem-no-wait}}$) can be determined by using equation 1.

$$T_{\text{mem-no-wait}} = 2\text{CLK} - T_{\text{addrset}} - T_{\text{dataset}} \quad (1)$$

where: $T_{\text{mem-no-wait}}$ = Memory access time for no wait state operation

2CLK = Two system clock (CLK) cycles

T_{addrset} = Maximum delay to valid address
+ Maximum throughput delay of address latch
+ Maximum delay to generate chip select
+ Maximum delay to generate $\overline{\text{SRAM-OE}}_n$

T_{dataset} = Maximum delay through data transceiver
+ Maximum data setup time of CPU

A similar analysis can be done for burst transactions. Equation 1 can be used to determine the access time for no wait state operation of the first word. For subsequent words, equation 2 can be used. In this equation, the address setup time is replaced by delay in the burst logic to change the address (T_{burst}). In this case, the data transfer of each subsequent word must be completed in one system clock (CLK) cycle (no address state). The minimum access time for a no wait state operation ($T_{\text{mem-no-wait}}$) can be determined by using the lesser value of equation 1 or equation 2.

$$T_{\text{mem-no-wait}} = \text{CLK} - T_{\text{burst}} - T_{\text{dataset}} \quad (2)$$

where: $T_{\text{mem-no-wait}}$ = Memory access time for no wait state operation

CLK = One system clock (CLK) cycles

T_{burst} = Maximum delay to change the address

T_{dataset} = Maximum delay through data transceiver
+ Maximum data setup time of CPU

The memory access time can be extended by delaying the $\overline{\text{READY}}$ signal and adding wait states.

The timing analysis described for a SRAM read operation can be used for EPROM timings. If EPROMs are only used to store initialization programs, they are seldom accessed compared to memory devices used to store program data or instructions. Consequently, the addition of wait states during the read cycle does not affect overall system performance.

Figure 4-6 shows the critical timing path for an SRAM write operation. The diagram consists of two time periods: the address setup period (T_{addrset}) and the memory response period (T_{mem}).

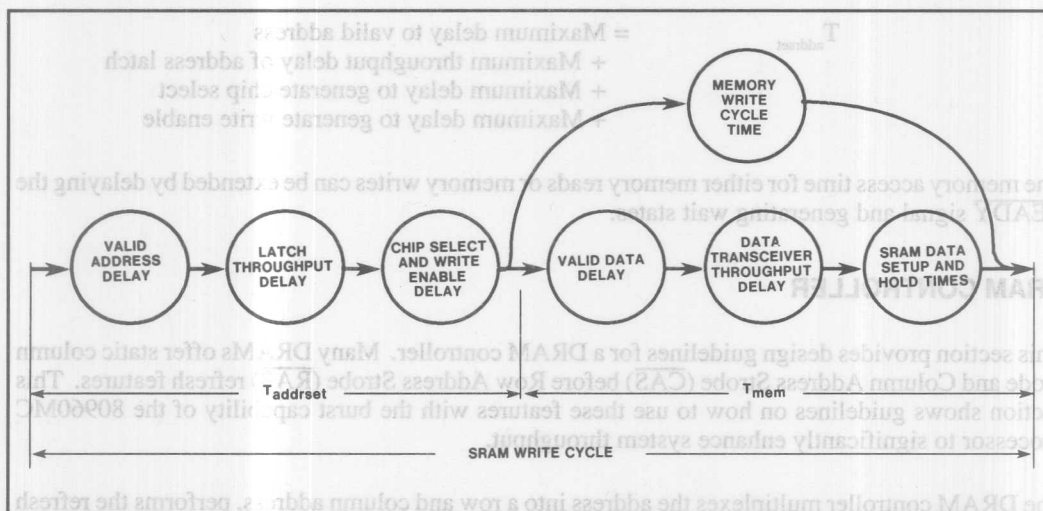


Figure 4-6: Critical Timing Path for SRAM Write Transaction

During the T_{addrset} period, the 80960MC processor outputs a valid address that is latched on the low-to-high transition of $\overline{\text{ALE}}$. The address decoder generates the $\overline{\text{SRAM-CS}}$ signal from the latched address and the Timing Control/SRAM Interface logic subsequently generates the Write Enable (WE) signal.

During the T_{mem} period the SRAM responds to the commands and writes the data. The access time of the memory determines the duration of the T_{mem} period. T_{mem} can be varied in increments of clock cycles by delaying the \overline{READY} signal.

Two timing paths should be considered during the T_{mem} period: the path where data is supplied to the memory, and the path that monitors the memory write cycle time. The first path takes into account the time for the 80960MC processor to generate valid data, the throughput delay of a data transceiver, and the data setup and hold time requirements of the memory devices. The second path is the memory write cycle specification. The longer of the two paths is the critical timing path.

By examining the timing path required to operate the SRAM, equation 2 can be derived which determines SRAM write cycle time for no wait state operation. The memory cycle time is determined by the lesser value of equation 1 or equation 2.

$$T_{mem-no-wait} = 2CLK - T_{addrset} \quad (3)$$

where: $T_{mem-no-wait}$ = \Rightarrow Maximum delay to valid data
+ Maximum throughput delay of data transceiver
+ Maximum data setup and hold times of memory
2CLK = Two system clock (CLK) cycles

$T_{addrset}$ = Maximum delay to valid address
+ Maximum throughput delay of address latch
+ Maximum delay to generate chip select
+ Maximum delay to generate write enable

The memory access time for either memory reads or memory writes can be extended by delaying the \overline{READY} signal and generating wait states.

DRAM CONTROLLER

This section provides design guidelines for a DRAM controller. Many DRAMs offer static column mode and Column Address Strobe (\overline{CAS}) before Row Address Strobe (\overline{RAS}) refresh features. This section shows guidelines on how to use these features with the burst capability of the 80960MC processor to significantly enhance system throughput.

The DRAM controller multiplexes the address into a row and column address, performs the refresh operation, arbitrates between a refresh request and memory request, and generates the necessary control signals for the DRAM. To implement these functions, the memory controller uses an address multiplexer, arbiter, refresh interval timer, and DRAM timing and control as shown in Figure 4-7.

A standard DRAM controller can be used, but it typically degrades system performance.

Address Multiplexer

The address multiplexer divides the DRAM address into a row and column address. The proper selection of a row or column address is accomplished by the row/column select signal (ROW/COL) from the DRAM timing and control circuit.

Refresh Interval Timer

The refresh interval timer periodically generates a refresh request (REF-REQ) by counting enough bus cycles to equal the refresh interval period. Since a refresh request is processed after a completed operation, the refresh period must take into account the time required to perform a bus operation, as well as the DRAM refresh specification. For example, a 1M-bit DRAM that requires 512 refresh cycles within 8 ms needs a refresh cycle every 15.6 μ s. To meet the DRAM specification, the refresh interval timer must generate a refresh request in less than 15.6 μ s to compensate for any required time to complete the operation with wait states.

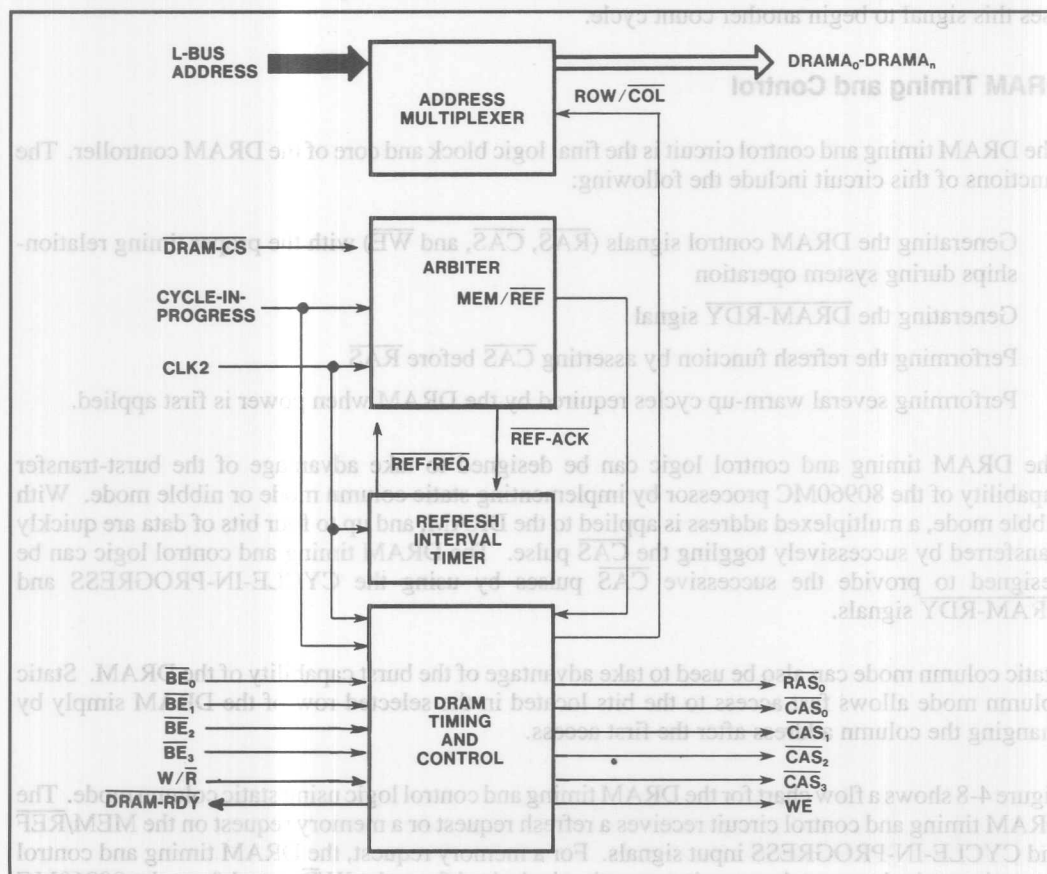


Figure 4-7: DRAM Controller Block Diagram

After the $\overline{\text{REF-REQ}}$ signal is generated, the arbiter sends a refresh acknowledge signal $\overline{\text{REF-ACK}}$ back to the interval timer to assure that refresh occurred before generating another $\overline{\text{REF-REQ}}$.

Arbiter

DRAM controller uses an arbiter to decide whether a memory cycle or refresh cycle is performed. In a synchronous design, arbitration is easily performed because memory and refresh cycle requests never occur at or near the same time.

The arbiter monitors memory cycle requests and refresh requests. The arbiter detects a DRAM memory request by decoding two signals: $\overline{\text{DRAM-CS}}$ and CYCLE-IN-PROGRESS . The $\overline{\text{REF-REQ}}$ signal indicates that a refresh cycle must be performed. The arbiter arbitrates between a memory cycle or refresh cycle and generates a Memory/Refresh (MEM/REF) signal. The DRAM timing and control block uses the MEM/REF signal to start the generation of the control signals.

When a refresh cycle is performed, the arbiter sends a $\overline{\text{REF-ACK}}$ signal to the refresh timer, which uses this signal to begin another count cycle.

DRAM Timing and Control

The DRAM timing and control circuit is the final logic block and core of the DRAM controller. The functions of this circuit include the following:

- Generating the DRAM control signals ($\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$) with the proper timing relationships during system operation
- Generating the $\overline{\text{DRAM-RDY}}$ signal
- Performing the refresh function by asserting $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$
- Performing several warm-up cycles required by the DRAM when power is first applied.

The DRAM timing and control logic can be designed to take advantage of the burst-transfer capability of the 80960MC processor by implementing static column mode or nibble mode. With nibble mode, a multiplexed address is applied to the DRAM, and up to four bits of data are quickly transferred by successively toggling the $\overline{\text{CAS}}$ pulse. The DRAM timing and control logic can be designed to provide the successive $\overline{\text{CAS}}$ pulses by using the CYCLE-IN-PROGRESS and $\overline{\text{DRAM-RDY}}$ signals.

Static column mode can also be used to take advantage of the burst capability of the DRAM. Static column mode allows fast access to the bits located in the selected row of the DRAM simply by changing the column address after the first access.

Figure 4-8 shows a flow chart for the DRAM timing and control logic using static column mode. The DRAM timing and control circuit receives a refresh request or a memory request on the MEM/REF and CYCLE-IN-PROGRESS input signals. For a memory request, the DRAM timing and control determines whether a read or a write operation is desired from the W/R signal from the 80960MC processor.

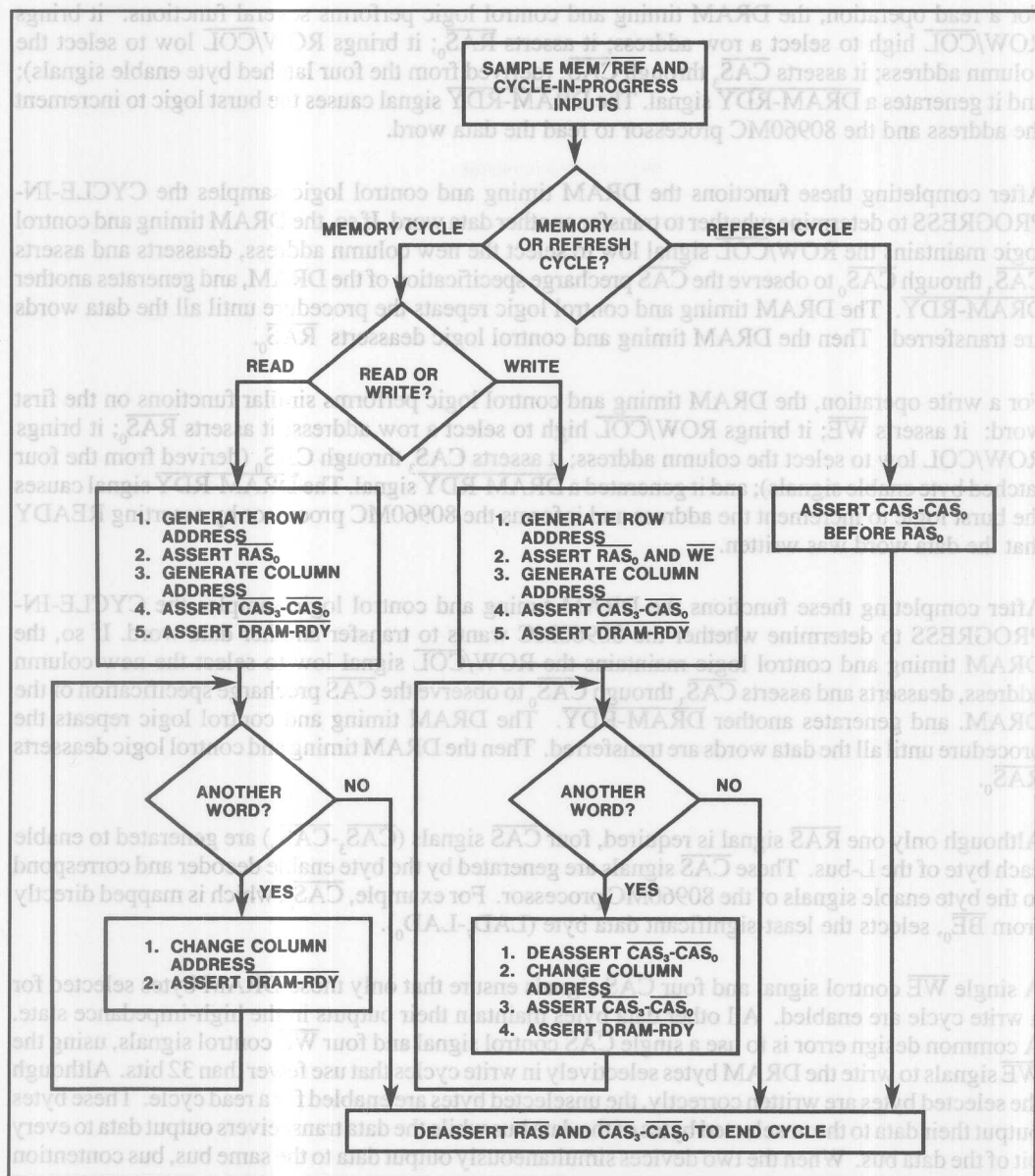


Figure 4-8: Flow Chart For DRAM Timing and Control Logic

The refresh function can be performed by asserting the $\overline{\text{CAS}}$ signal before asserting $\overline{\text{RAS}}$. The $\overline{\text{CAS}}$ pulse is activated prior to the assertion of the $\overline{\text{RAS}}$ pulse, the DRAM automatically performs a refresh cycle on one row by employing an on-chip address counter. Upon completion of the refresh

For a read operation, the DRAM timing and control logic performs several functions: it brings ROW/COL high to select a row address; it asserts \overline{RAS}_0 ; it brings ROW/COL low to select the column address; it asserts \overline{CAS}_3 through \overline{CAS}_0 (derived from the four latched byte enable signals); and it generates a $\overline{DRAM-RDY}$ signal. The $\overline{DRAM-RDY}$ signal causes the burst logic to increment the address and the 80960MC processor to read the data word.

After completing these functions the DRAM timing and control logic samples the CYCLE-IN-PROGRESS to determine whether to transfer another data word. If so, the DRAM timing and control logic maintains the ROW/COL signal low to select the new column address, deasserts and asserts \overline{CAS}_3 through \overline{CAS}_0 to observe the \overline{CAS} precharge specification of the DRAM, and generates another $\overline{DRAM-RDY}$. The DRAM timing and control logic repeats the procedure until all the data words are transferred. Then the DRAM timing and control logic deasserts \overline{RAS}_0 .

For a write operation, the DRAM timing and control logic performs similar functions on the first word: it asserts \overline{WE} ; it brings ROW/COL high to select a row address; it asserts \overline{RAS}_0 ; it brings ROW/COL low to select the column address; it asserts \overline{CAS}_3 through \overline{CAS}_0 (derived from the four latched byte enable signals); and it generated a $\overline{DRAM-RDY}$ signal. The $\overline{DRAM-RDY}$ signal causes the burst logic to increment the address and informs the 80960MC processor by asserting READY that the data word was written.

After completing these functions the DRAM timing and control logic samples the CYCLE-IN-PROGRESS to determine whether the 80960MC wants to transfer another data word. If so, the DRAM timing and control logic maintains the ROW/COL signal low to select the new column address, deasserts and asserts \overline{CAS}_3 through \overline{CAS}_0 to observe the \overline{CAS} precharge specification of the DRAM, and generates another $\overline{DRAM-RDY}$. The DRAM timing and control logic repeats the procedure until all the data words are transferred. Then the DRAM timing and control logic deasserts \overline{RAS}_0 .

Although only one \overline{RAS} signal is required, four \overline{CAS} signals (\overline{CAS}_3 - \overline{CAS}_0) are generated to enable each byte of the L-bus. These \overline{CAS} signals are generated by the byte enable decoder and correspond to the byte enable signals of the 80960MC processor. For example, \overline{CAS}_0 , which is mapped directly from \overline{BE}_0 , selects the least-significant data byte (LAD₇-LAD₀).

A single \overline{WE} control signal and four \overline{CAS} signals ensure that only those DRAM bytes selected for a write cycle are enabled. All other data bytes maintain their outputs in the high-impedance state. A common design error is to use a single \overline{CAS} control signal and four \overline{WE} control signals, using the \overline{WE} signals to write the DRAM bytes selectively in write cycles that use fewer than 32 bits. Although the selected bytes are written correctly, the unselected bytes are enabled for a read cycle. These bytes output their data to the unselected bytes of the data bus while the data transceivers output data to every bit of the data bus. When the two devices simultaneously output data to the same bus, bus contention occurs.

The refresh function can be performed by asserting the \overline{CAS} signal before asserting \overline{RAS} . The \overline{CAS} before \overline{RAS} refresh feature eliminates the need for an external refresh address counter. When the \overline{CAS} pulse is activated prior to the assertion of the \overline{RAS} pulse, the DRAM automatically performs a refresh cycle on one row by employing an on-chip address counter. Upon completion of the refresh

cycle, the address counter is automatically incremented. The $\overline{\text{MEM/REF}}$ signal from the arbiter can be used by the DRAM timing and control logic block to initiate a $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh cycle.

Besides generating the $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{WE}}$ signals, the DRAM timing and control logic generates a number of warm-up cycles for the DRAM after reset by issuing several refresh requests.

Timing Considerations For The DRAM Controller

Figure 4-9 shows a typical example of a timing diagram for a two-word read transaction that uses static column mode; similarly, Figure 4-10 is a typical example for a two-word write transaction. The example assumes a memory access time that requires two wait states (T_w) for the initial data word and one wait for the second data word.

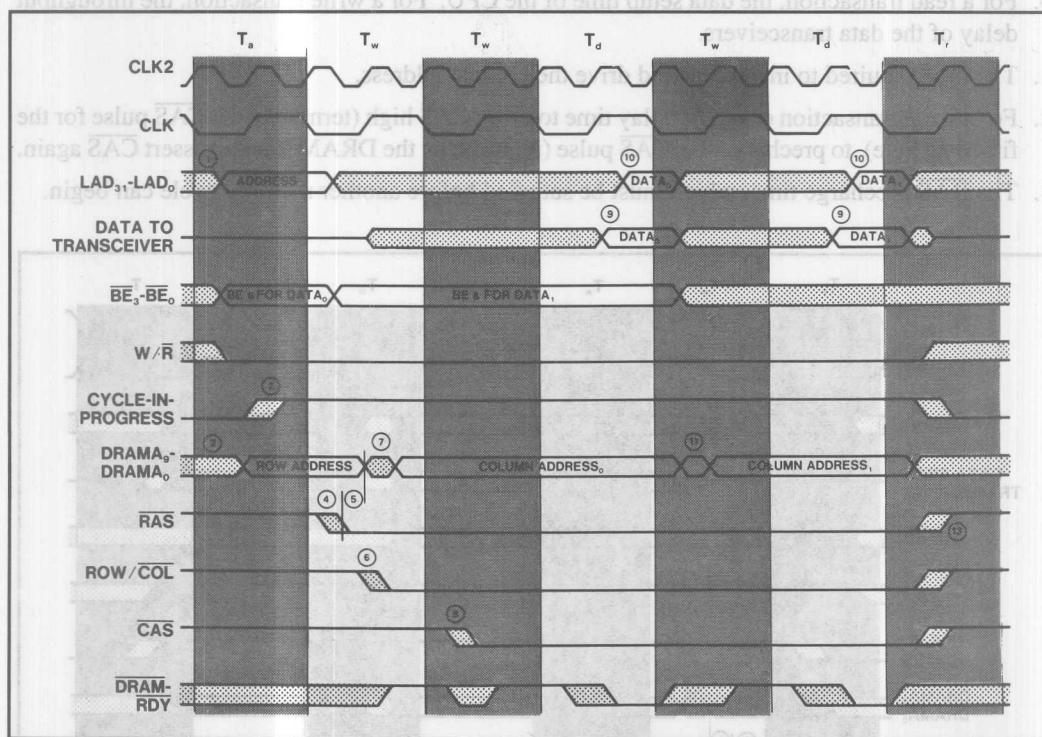


Figure 4-9: Timing Diagram for Two-word Read Transaction

The critical timing areas for both read and write transactions are noted by circled numbers in the diagrams, which are enumerated below.

1. The delay for the CPU to generate a valid address.
2. The delay for the DRAM timing and control logic to generate the CYCLE-IN-PROGRESS signal.

3. The delay to generate the DRAM row address. This time includes the address latch throughput delay, the multiplexer throughput delay, and the address driver delay.
4. The delay to generate $\overline{\text{RAS}}$, which includes the delay to generate the $\overline{\text{DRAM-CS}}$ signal.
5. The row address hold time after the high-to-low transition of $\overline{\text{RAS}}$.
6. The time required to generate the multiplexer control signal ($\text{ROW}/\overline{\text{COL}}$) after the row address hold time is satisfied.
7. The time required to switch from a row to column address plus any driver delays.
8. The delay to generate and drive the $\overline{\text{CAS}}$ signals.
9. For a read transaction, the throughput delay of the data transceivers. For a write transaction, the delay by the CPU to generate valid data.
10. For a read transaction, the data setup time of the CPU. For a write transaction, the throughput delay of the data transceivers.
11. The time required to increment and drive the column address.
12. For a write transaction only, the delay time to bring $\overline{\text{CAS}}$ high (terminate the $\overline{\text{CAS}}$ pulse for the first data byte), to precharge the $\overline{\text{CAS}}$ pulse (required by the DRAM), and to assert $\overline{\text{CAS}}$ again.
13. The $\overline{\text{RAS}}$ precharge time, which must be satisfied before another memory cycle can begin.

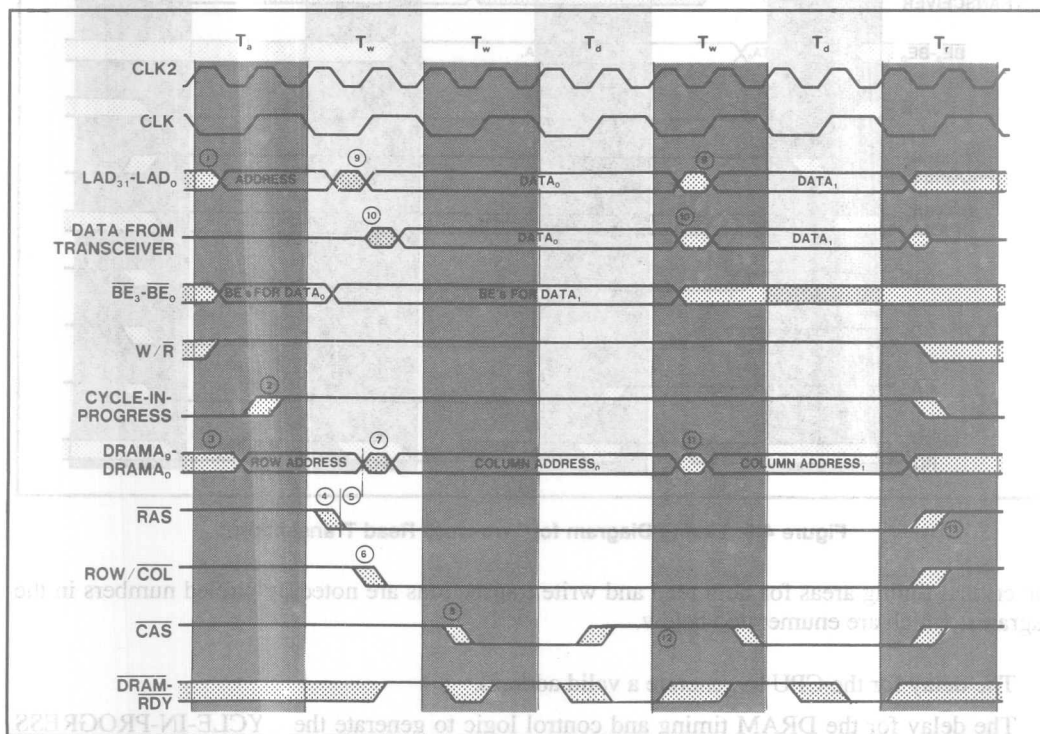


Figure 4-10: Timing Diagram for Two-word DRAM Write Transaction

DRAM Interleaving

Because the DRAM consists of dynamic nodes, a row precharge time is required to recharge the nodes after every memory cycle. This time must be included in the timing evaluation, as noted by the example. To avoid the precharge time delay of the DRAM, the memory array can be arranged so that each subsequent memory access is most likely to be directed to a different bank. In this configuration, wait time between accesses is not required because while one bank of DRAMs performs the current access, another bank precharges and is ready to perform the next access immediately.

If DRAMs are interleaved (i.e., arranged in multiple banks so that adjacent addresses are in different banks), the DRAM precharge time can be masked for most accesses. With two banks of DRAMs, one for even 32-bit addresses and one for odd 32-bit addresses, all sequential 32-bit accesses can be completed without waiting for the DRAM to precharge.

Even when random accesses are made, two DRAM banks allow 50 percent of back-to-back accesses to be made without waiting for the DRAMs to precharge. The precharge time is also masked when the 80960MC processor has no bus accesses to be performed. During these idle bus cycles, the most recently accessed DRAM bank can precharge so that the next memory access to either bank can begin immediately.

SUMMARY

The memory interface circuit allows the 80960MC processor to communicate with the memory devices. The basic memory interface logic can be divided into six blocks: the data transceivers, the address latches, the address decoder, the burst logic, the DRAM timing and control logic, and the byte enable latch. The DRAM controller and SRAM interface complete the memory interface circuit. The DRAM controller can be designed to take advantage of the 80960MC processor's burst capability to enhance system performance.

This chapter focused on the design guidelines for the memory interface design to the 80960MC processor. Chapter 5 develops guidelines on designing peripheral devices in the single-processor hardware system.

NO Interface

2

CHAPTER 5 I/O INTERFACE

The 80960MC processor supports 8, 16, 32-bit I/O devices by mapping them into its 4 G-byte memory address space. This chapter describes the design considerations for the interface between the 80960MC processor and I/O components. Several examples illustrate the design concepts.

INTERFACING TO 8-BIT AND 16-BIT PERIPHERALS

The 80960MC processor accesses I/O devices by using a memory-mapped address. Consequently, memory-type instructions can be used to perform input/output operations. For example, the 80960MC processor's LOAD and STORE instructions can directly support 8-bit and 16-bit data moves to or from I/O peripherals. The instructions include those listed below.

- Load Ordinal Byte (reads a byte)
- Load Ordinal Short (reads 16-bit data)
- Store Ordinal Byte (writes a byte)
- Store Ordinal Short (writes 16-bit data)

These instructions perform the transfer on the data bits specified by the two low-order lines of the effective address. See the *80960MC Programmer's Reference Manual* for complete details.

GENERAL SYSTEM INTERFACE

In a typical 80960MC processor system design, a number of slave I/O devices can be controlled through a general system interface. Other I/O devices, particularly those capable of controlling the L-bus, can use the general system interface, but may require additional logic to isolate the bus. This section describes the general system interface and assumes that the 80960MC processor does not perform burst transactions to the I/O devices.

Figure 5-1 shows the major logic blocks of the general system interface. Standard 8-bit data transceivers add drive capability, provide bus isolation, and prevent bus conflicts that may occur with slow I/O components. The address latch demultiplexes the address/data lines and holds the address stable throughout the L-bus transaction. The address decoder generates the I/O chip-select signals from the latched address lines. The timing control block provides the READY signal to the 80960MC processor and the I/O read and I/O write command.

This basic interface circuit is quite similar to the one used in the basic memory interface described in Chapter 4. For most systems the same data transceivers, address decoders, and address latches can be used to access both memory and I/O devices. The timing control logic can be implemented to accommodate both memory and I/O devices.

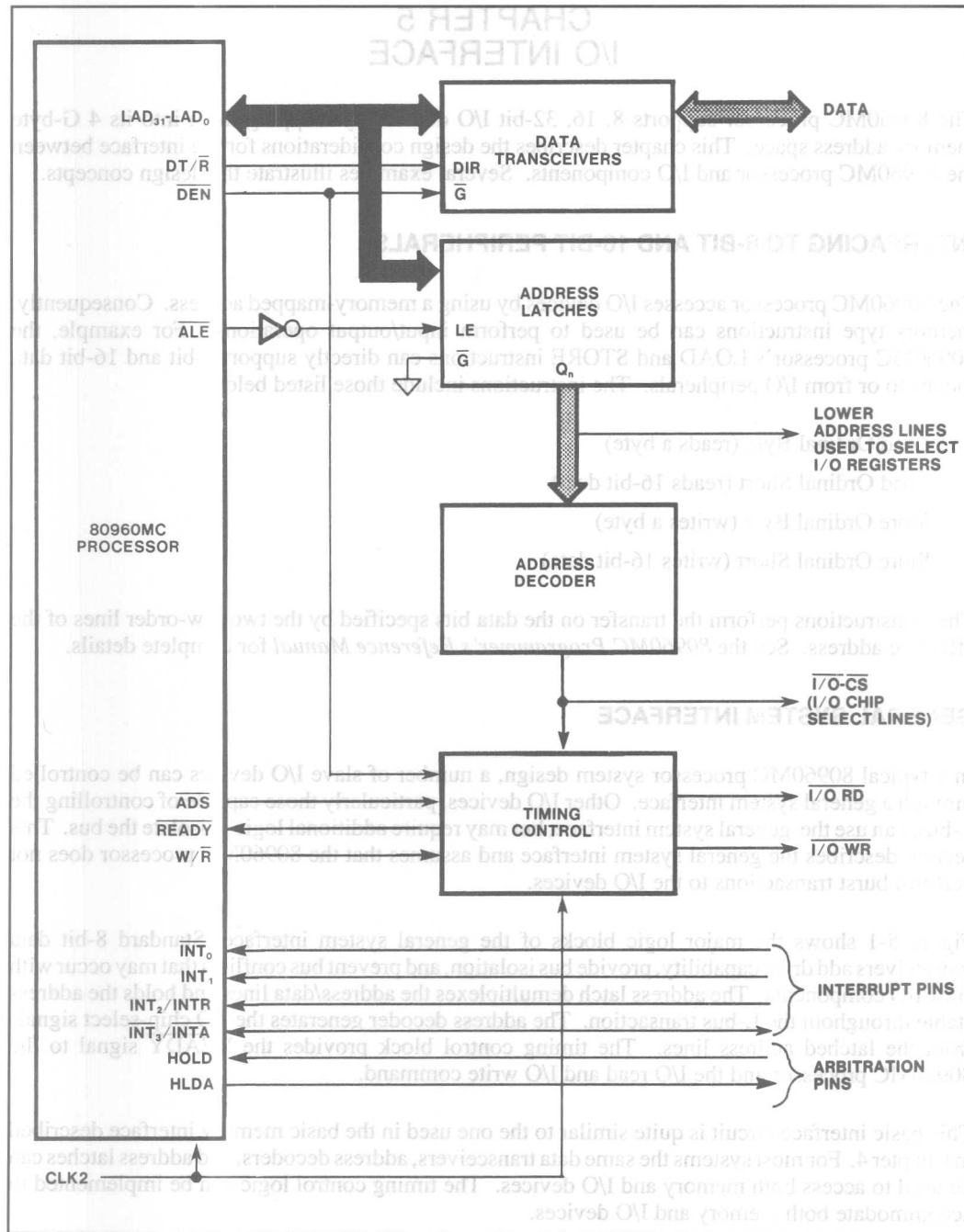


Figure 5-1: Simplified I/O Interface

Data Transceivers

Standard 8-bit transceivers can be used to provide isolation and additional drive capability for the L-bus. Transceivers prevent bus contention that can occur if some devices are slow to remove data from the data bus after a read cycle. For example, if an I/O write cycle follows a I/O read cycle, the 80960MC processor may drive the L-bus before a slow device has removed its outputs from the bus, potentially causing a current spike. Transceivers, however, can be omitted if the data float time of the device is short enough and the load does not exceed the 80960MC device specifications.

The data transceiver can be controlled by two signals from the 80960MC processor: Data Transmit/Receive ($\overline{DT/R}$) and Data Enable (\overline{DEN}). $\overline{DT/R}$ indicates the direction of data flow and \overline{DEN} enables the transceivers.

Address Latch/Demultiplexer

Standard transparent latches can be used to demultiplex the address/data lines of the 80960MC processor. The latch is controlled by the \overline{ALE} signal from the 80960MC processor. The \overline{ALE} signal passes through an inverter, such that when \overline{ALE} goes low, the address flows through the latch. The low-to-high transition of \overline{ALE} can be used to latch the address.

If only slave-type peripherals are used in a system, the output enable of the latches can always remain active by connecting it to ground. For systems with DMA devices, the output enable can be used to permit the DMA device to drive a common address bus.

Address Decoder

The address decoder determines which particular I/O device is selected by decoding the address. The I/O address can be any address in the 4 Gbyte address range except for the upper 16 Mbytes (addresses $FF000000_H$ through $FFFFFFFF_H$), which the 80960MC processor reserves for inter-agent communication and internal I/O. Typically, a small range of address bits are reserved for accessing I/O devices by defining certain higher-order address bits as an I/O access.

As an example, consider a 32-bit address: A_{31} through A_{15} could indicate an I/O access when A_{31} is set to zero, and A_{30} - A_{15} are set to one; A_{14} through A_5 could then be used to specify a particular I/O device; and A_4 through A_2 can be used to access up to 8 registers of the I/O component. A_1 and A_0 are not used by the I/O device. This particular scheme selects up to 1,024 devices, while using only 32K bytes of the available 4 Gbytes of address space.

The address decoder can be located either before or after the address latches. Usually, it is placed after the latches, so that the chip-select signal does not need an additional latch.

Timing Control Logic

The timing control logic accommodates I/O devices that cannot transfer information at the maximum bus rate by inserting Wait States until the data becomes available. The timing control logic consists

of a counter and timing logic, as shown in Figure 5-2. The counter produces a 4-bit binary count. The count is started at the beginning of the operation (determined by $\overline{\text{ADS}}$ and $\overline{\text{DEN}}$) and is stopped by the $\overline{\text{READY}}$ signal. The timing logic asserts the $\overline{\text{READY}}$ signal, the I/O write command ($\overline{\text{I/O-WR}}$), and the I/O read command ($\overline{\text{I/O-RD}}$) based upon the clock count, the I/O chip select signal ($\overline{\text{I/O-CS}}$), and the W/R command.

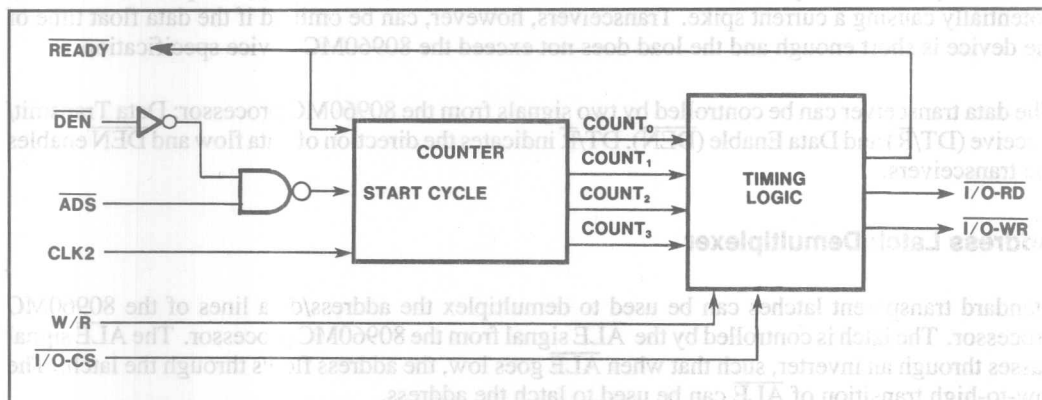


Figure 5-2: I/O Timing Control Block Diagram

For many peripherals, the timing logic can be programmed to assert $\overline{\text{READY}}$ at the appropriate count for the selected device. Specific I/O chip select signals can be used to indicate how many clock cycles to wait before asserting $\overline{\text{READY}}$.

For some I/O peripherals, particularly bus masters, $\overline{\text{READY}}$ cannot be determined by counting clock cycles. For these I/O devices, $\overline{\text{READY}}$ can be supplied by the device and passed on to 80960MC processor.

The timing control block can assert the $\overline{\text{I/O-RD}}$ or $\overline{\text{I/O-WR}}$ signal for I/O devices based upon the clock count. The timing for these signals can be selected for the slowest device to simplify the logic circuit or can be customized for each individual peripheral device to maximize performance.

I/O INTERFACE DESIGN EXAMPLES

The general system interface shown in Figure 5-1 can be used to connect the 80960MC processor to many slave peripherals. The following list includes some common peripherals compatible with this interface:

- M8259A Programmable Interrupt Controller
- M8253, M8254 Programmable Interval Timer
- M82510, Asynchronous Serial Controller
- M8274 Multi-Protocol Serial Controller

- M8255 Programmable Peripheral Interface
- 82586 LAN Coprocessor (not offered in a MIL-STD-883C version)
- M82786 Graphics Coprocessor

This section provides guidelines and design considerations for interfacing the 80960MC processor to different types of I/O configurations. Specifically, three design examples are examined. The M8259A design example shows how to interface the 80960MC processor to a slave-type peripheral device. The 82586 design example shows how a 16-bit bus master reads and writes to the 80960MC processor's system memory. The M82786 design example shows how the 80960MC processor can read or write to graphics memory using a 16-bit data bus.

M8259A Programmable Interrupt Controller

The M8259A Programmable Interrupt Controller is designed for use in interrupt-driven microcomputer systems, where it manages up to eight independent interrupts. The M8259A handles interrupt priority resolution and returns an 8-bit vector to the 80960MC processor during an interrupt-acknowledge cycle. Intel *Application Note AP-59* contains detailed information on configurations of the M8259A.

Interface

Figure 5-3 shows the connection of the 80960MC processor to a single M8259A Interrupt Controller. This circuit consists of the general system interface plus a bidirectional buffer. The example assumes that several interrupt requests occur at the same time so that priority resolution is required.

The data lines from the M8259A are not directly aligned to the 80960MC processor because of the difference in priority resolution between the devices. Although both devices use an 8-bit interrupt vector, the 80960MC processor implicitly defines the priority by dividing the interrupt vector by eight. The M8259A defines the priority in the lower three bits of the interrupt vector. Furthermore, the highest priority vector of the 80960MC processor has a value of 31 in the upper five bits of the interrupt vector. Whereas, the highest priority interrupt of the M8259A has a value of 0 in the lower three bits of the interrupt vector.

To resolve the priority difference, the interrupt vector from the M8259A can be inverted and rotated left by three bits as shown by the data alignment between the 80960MC processor and M8259A in Figure 5-3. Rotating the data bits in this manner provides two advantages: the interrupt table for the M8259A can be located by contiguous addresses, and the upper two most significant bits of the interrupt vector remain free to group interrupt vectors if additional M8259As are needed.

Care must be exercised, however, when programming the registers of the M8259A. For example, assume that the second initialization command word (ICW2 register) of the M8259A requires a data byte value of 00011111_2 . To transfer the correct information, the 80960MC processor needs to write a data word with the value of 00000111_2 because this word is rotated left three places and inverted.

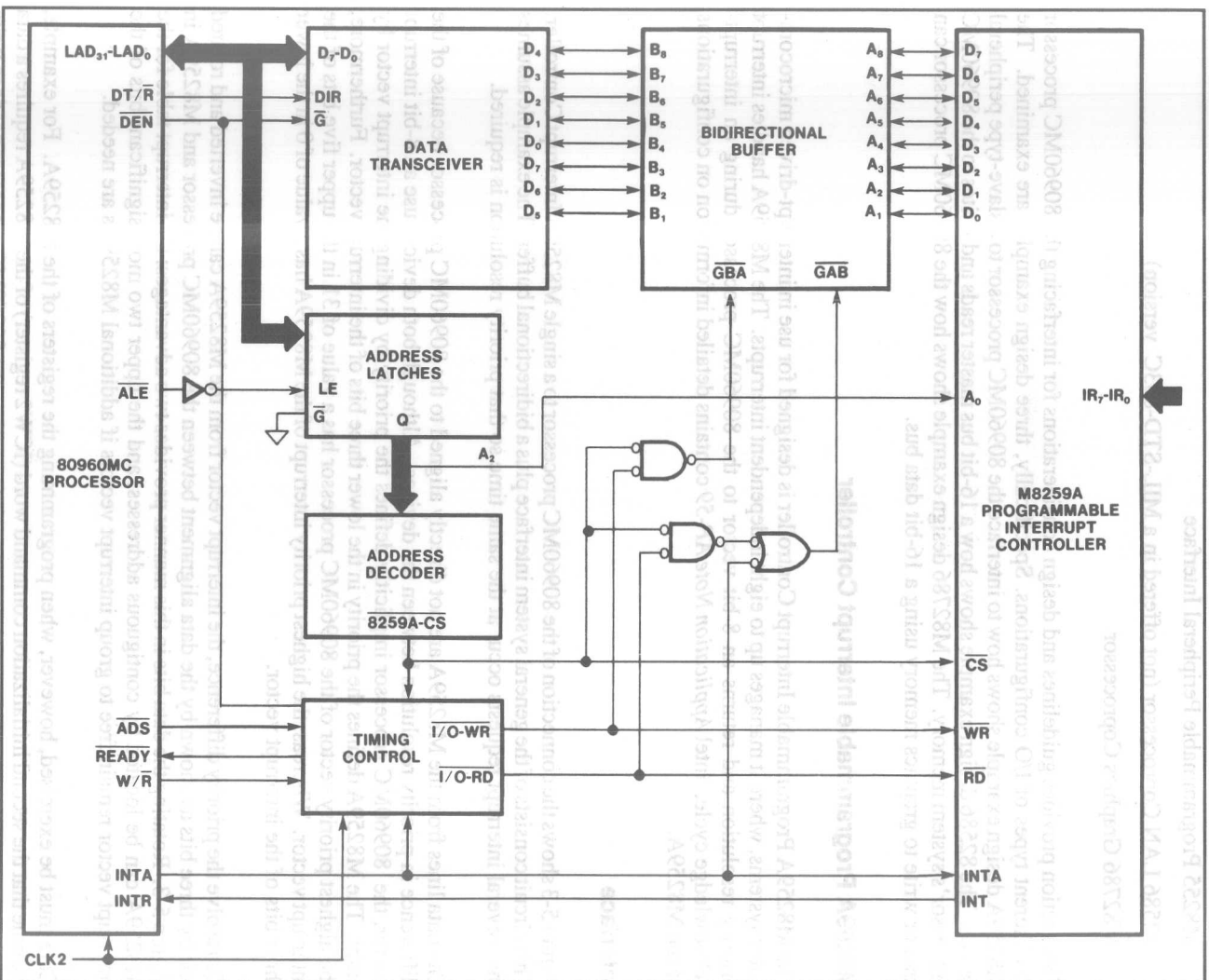


Figure 5-3: Block Diagram for M8259A Interface

Operation

The M8259A starts the interrupt cycle by generating an interrupt request (INT) to the 80960MC processor, which receives the signal at the INTR input pin. This assumes the Interrupt Control register of the 80960MC processor is set to accommodate an external interrupt controller.

When the 80960MC processor comes to a breakpoint in its execution, it asserts the $\overline{\text{INTA}}$ signal twice. The first $\overline{\text{INTA}}$ signal acknowledges the interrupt request and causes the M8259A to prioritize the interrupt requests it received up to this point. The $\overline{\text{INTA}}$, together with the $\overline{\text{M8259A-CS}}$, are applied to the timing control logic to generate a $\overline{\text{READY}}$ signal.

The 80960MC processor automatically asserts the second $\overline{\text{INTA}}$ signal five clock cycles after the assertion of $\overline{\text{READY}}$. After the second assertion of $\overline{\text{INTA}}$, the 80960MC processor reads the interrupt vector from the M8259A.

The bidirectional buffer inverts and passes the 8-bit vector to the 80960MC processor with the appropriate lines rearranged. The output enable signal for the data buffer is controlled by $\overline{\text{INTA}}$ for this operation. After the data transfer is completed, the timing control circuit generates a second $\overline{\text{READY}}$ signal to terminate the interrupt acknowledge cycle.

The same circuitry can be used to read or write to the M8259A registers. In this case, the 80960MC processor selects the M8259A through a memory-mapped address. Local address line 2 (A_2) selects one of two internal registers of the M8259A. The I/O read or I/O write command is generated by the timing control circuit. The data passes through the bidirectional data buffer to or from the selected register of the M8259A.

The direction of data flow through the buffer is controlled by three logic gates shown in Figure 5-3. For an I/O write operation, the $\overline{\text{I/O Write}}$ command and $\overline{\text{M8259A-CS}}$ signal control the output enable signal of the bidirectional buffer. Similarly, for a read operation, the $\overline{\text{I/O Read}}$ command and the $\overline{\text{M8259A-CS}}$ signal control the output enable signal of the buffer. After the data is transferred, the timing control circuit asserts $\overline{\text{READY}}$.

82586 Local Area Network Coprocessor Example

The 82586 (not offered in a MIL-STD-883C version) is an intelligent, high-performance communications controller designed to perform most tasks required for controlling access to a local area network (LAN), such as Ethernet or Starlan. In many applications, the 82586 is the communication manager for a station connected to a LAN controller. Such a station usually includes a host CPU, shared memory, a Serial Interface Unit, a transceiver, and LAN controller link, as shown in Figure 5-4. The 82586 performs all functions associated with data transfer between the shared memory and the LAN link, including:

- Framing
- Link management
- Address filtering

- Error detection
- Data encoding
- Network management
- Direct memory access
- Buffer chaining
- High-level (user) command interpretation

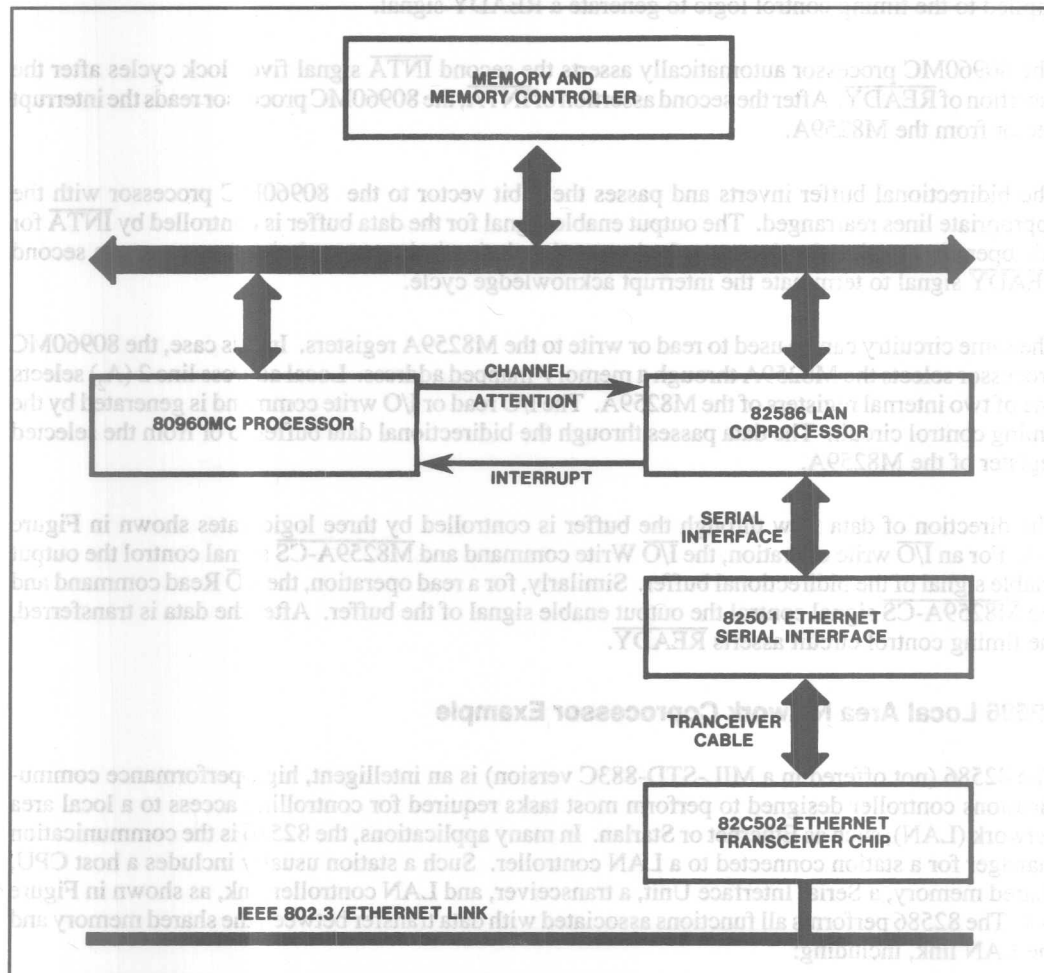


Figure 5-4: Lan Station

The 82586 has two interfaces: a 16-bit bus interface and a network interface to the Serial Interface Unit. The bus interface is described here. For detailed information on using the 82586, refer to the *Local Area Networking Component User's Manual*.

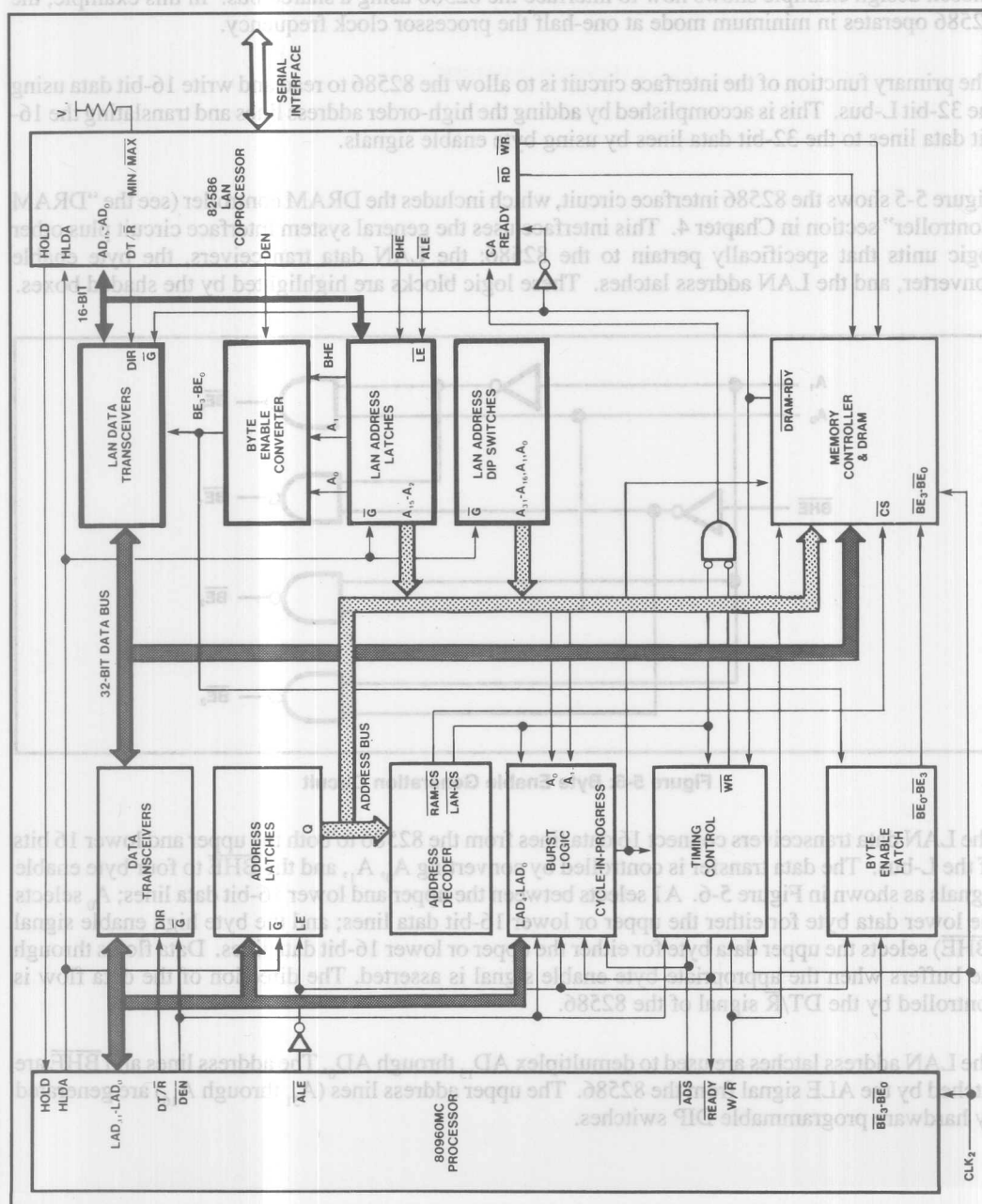


Figure 5-5: Block Diagram For LAN Controller Interface

Interface

There are several ways to design an interface between the 82586 and the 80960MC processor. The chosen design example shows how to interface the 82586 using a shared bus. In this example, the 82586 operates in minimum mode at one-half the processor clock frequency.

The primary function of the interface circuit is to allow the 82586 to read and write 16-bit data using the 32-bit L-bus. This is accomplished by adding the high-order address lines and translating the 16-bit data lines to the 32-bit data lines by using byte enable signals.

Figure 5-5 shows the 82586 interface circuit, which includes the DRAM controller (see the “DRAM Controller” section in Chapter 4. This interface uses the general system interface circuit plus other logic units that specifically pertain to the 82586: the LAN data transceivers, the byte enable converter, and the LAN address latches. These logic blocks are highlighted by the shaded boxes.

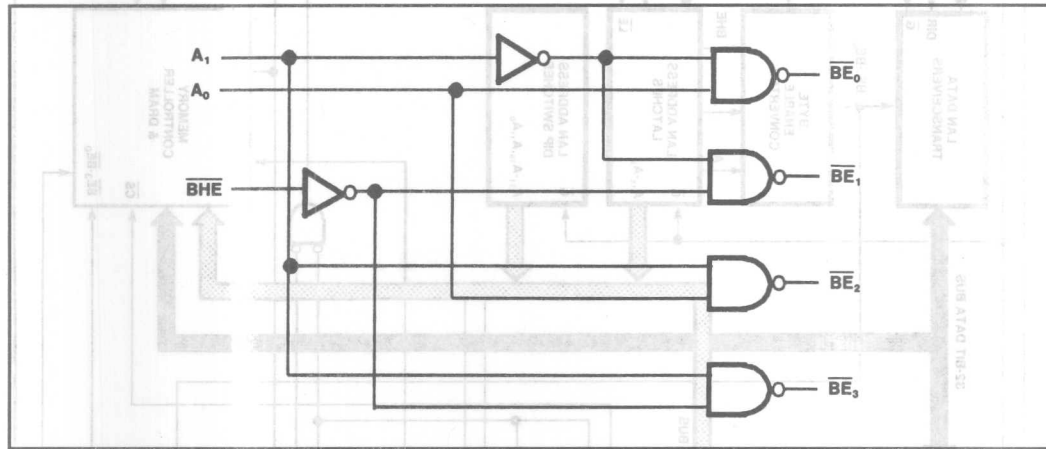


Figure 5-6: Byte Enable Generation Circuit

The LAN data transceivers connect 16 data lines from the 82586 to both the upper and lower 16 bits of the L-bus. The data transfer is controlled by converting A_0 , A_1 , and the \overline{BHE} to four byte enable signals as shown in Figure 5-6. A_1 selects between the upper and lower 16-bit data lines; A_0 selects the lower data byte for either the upper or lower 16-bit data lines; and the byte high enable signal (\overline{BHE}) selects the upper data byte for either the upper or lower 16-bit data lines. Data flows through the buffers when the appropriate byte enable signal is asserted. The direction of the data flow is controlled by the DT/\overline{R} signal of the 82586.

The LAN address latches are used to demultiplex AD_{15} through AD_0 . The address lines and \overline{BHE} are latched by the ALE signal from the 82586. The upper address lines (A_{31} through A_{16}) are generated by hardware programmable DIP switches.

The 82586 begins operation when the Channel Attention (CA) input signal is asserted. This signal is generated by gating the write command and 82586 chip select signal.

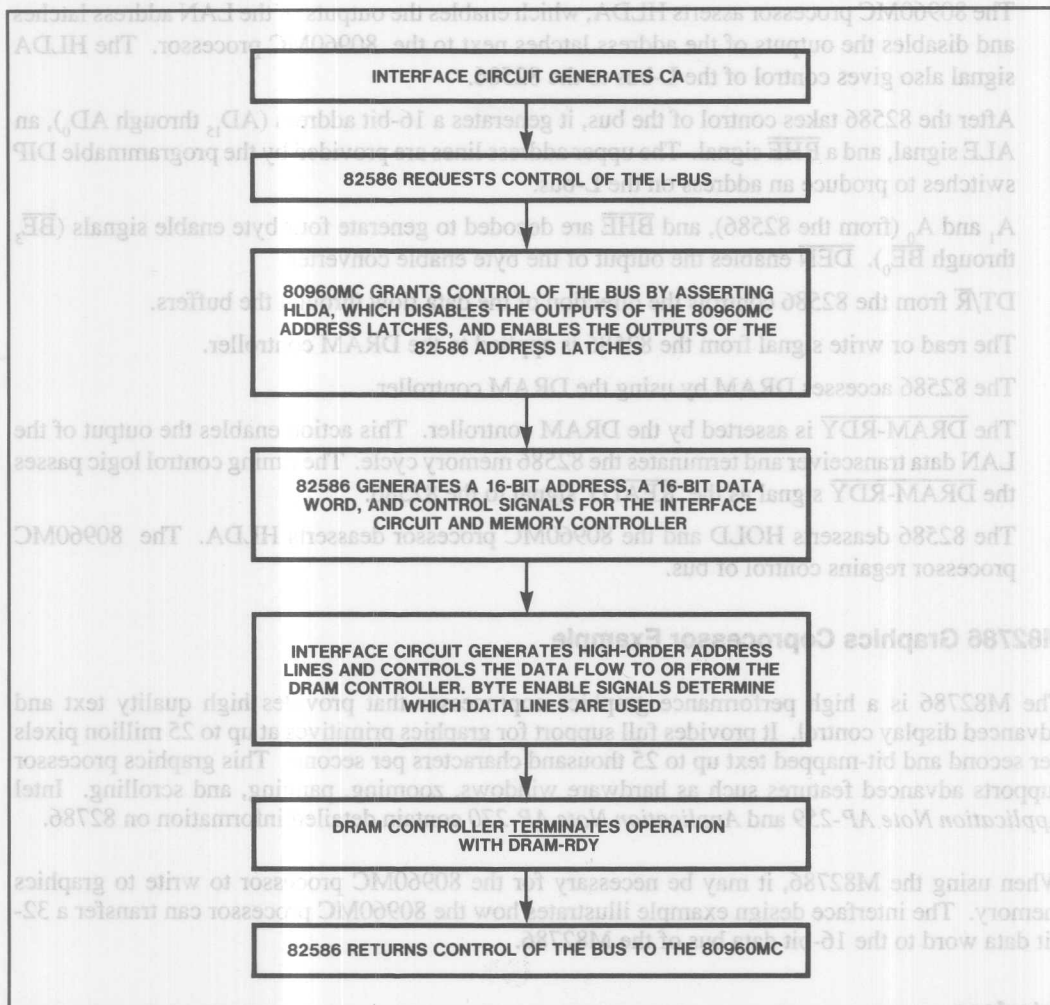


Figure 5-7: Operational Flow Diagram For 82586 Interface

Operation

The interaction between the 82586 and the 80960MC processor is described below and is summarized in Figure 5-7.

- The 80960MC processor invokes the 82586 by supplying a memory-mapped address and a write command. The memory-mapped address results in a 82586-CS signal, which is gated with a

write command to produce the CA signal.

- The 82586 responds by generating a hold request and waits for HLDA.
- The 80960MC processor asserts HLDA, which enables the outputs of the LAN address latches and disables the outputs of the address latches next to the 80960MC processor. The HLDA signal also gives control of the L-bus to the 82586.
- After the 82586 takes control of the bus, it generates a 16-bit address (AD_{15} through AD_0), an ALE signal, and a \overline{BHE} signal. The upper address lines are provided by the programmable DIP switches to produce an address on the L-bus.
- A_1 and A_0 (from the 82586), and \overline{BHE} are decoded to generate four byte enable signals (\overline{BE}_3 through \overline{BE}_0). \overline{DEN} enables the output of the byte enable converter.
- DT/\overline{R} from the 82586 controls the direction of the data flow through the buffers.
- The read or write signal from the 82586 is applied to the DRAM controller.
- The 82586 accesses DRAM by using the DRAM controller.
- The $\overline{DRAM-RDY}$ is asserted by the DRAM controller. This action enables the output of the LAN data transceiver and terminates the 82586 memory cycle. The timing control logic passes the $\overline{DRAM-RDY}$ signal as the \overline{READY} signal to the 82586.
- The 82586 deasserts HOLD and the 80960MC processor deasserts HLDA. The 80960MC processor regains control of bus.

M82786 Graphics Coprocessor Example

The M82786 is a high performance graphics coprocessor that provides high quality text and advanced display control. It provides full support for graphics primitives at up to 25 million pixels per second and bit-mapped text up to 25 thousand characters per second. This graphics processor supports advanced features such as hardware windows, zooming, panning, and scrolling. Intel *Application Note AP-259* and *Application Note AP-270* contain detailed information on 82786.

When using the M82786, it may be necessary for the 80960MC processor to write to graphics memory. The interface design example illustrates how the 80960MC processor can transfer a 32-bit data word to the 16-bit data bus of the M82786.

Interface

There are several ways to design an interface between the M82786 and the 80960MC processor. In this example, the 80960MC processor reads or writes to graphics memory by accessing the M82786 through the interface logic circuit. This example assumes that the M82786 operates in the slave mode, and that the 80960MC processor does not perform burst transfers. The 80960MC processor only performs burst transfers for instructions that specify accesses for more than one word or for instruction fetches.

The interface circuit translates a 32-bit data bus to a 16-bit data bus by dividing the data lines into the upper and lower 16 bits and sequencing the data transmission. When the 80960MC processor writes to graphics memory, the bidirectional transceivers sequence the lower and the upper data bits of the L-bus to the 16-bit data bus of the M82786.

The process is reversed when the 80960MC processor reads from graphics memory. The bidirectional transceivers form a 32-bit data word by latching the first 16-bit data word on the lower data lines, routing the next 16 bits to the upper data lines, and then passing the 32-bit data word on the L-bus.

Figure 5-8 shows the details of the graphics controller interface circuit. This interface uses the general system interface circuit plus the following logic units: the bidirectional transceivers, the data buffer control, the data bus controller, and the address translator. These logic blocks are highlighted by the shaded boxes.

The bidirectional transceivers pass data to (from) a 32-bit data bus from (to) a 16-bit data bus. Data is sequenced through the transceivers by the control signals generated by the data buffer controller.

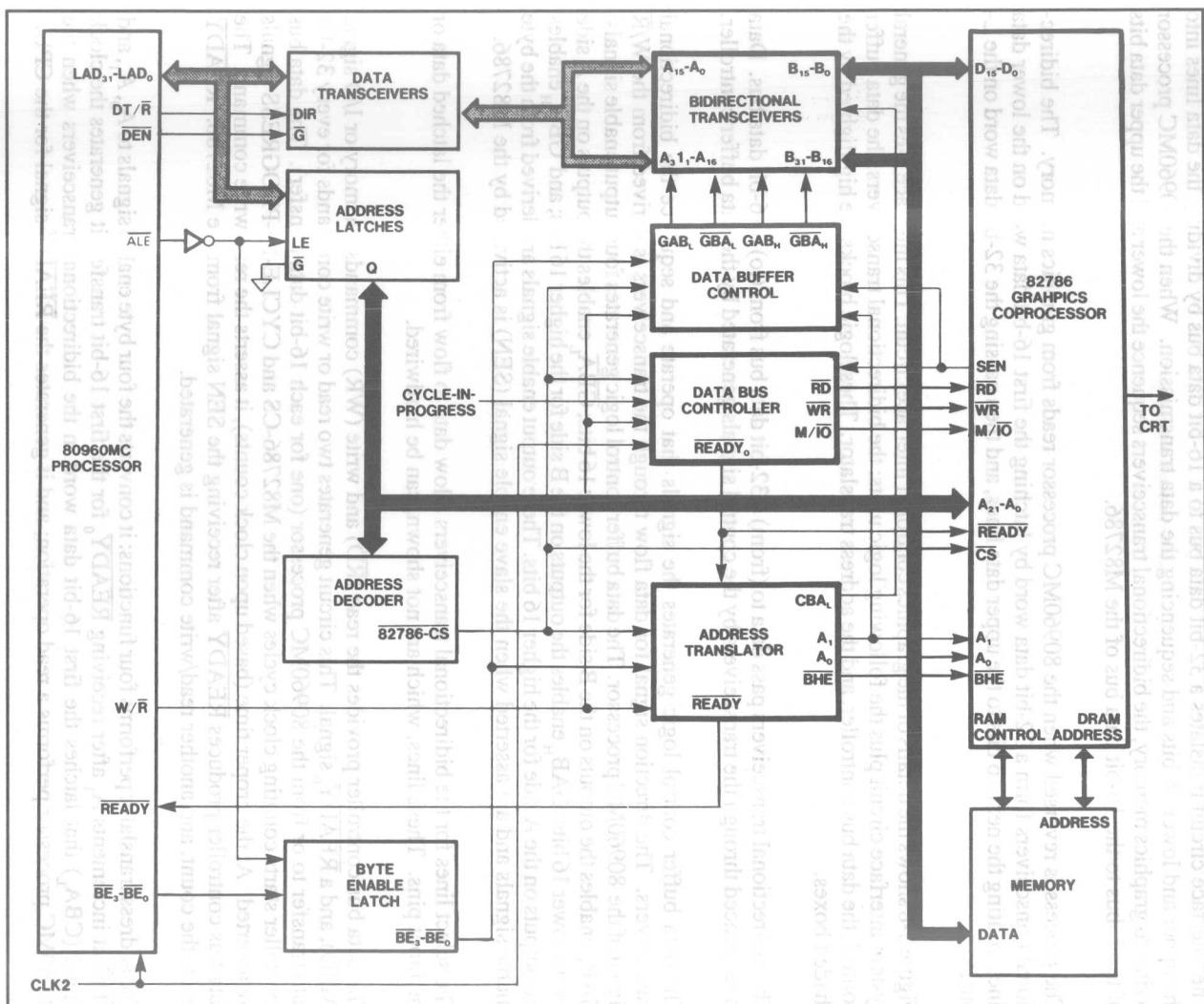
The data buffer control logic generates the signals that operate and sequence the bidirectional transceivers. The direction signal for data flow through the transceivers is derived from the $\overline{W}/\overline{R}$ signal of the 80960MC processor. The data buffer control logic generates four output enable signals: \overline{GAB}_L enables the outputs on the B side for the lower 16 bits; \overline{GBA}_L enables the outputs on the A side for the lower 16 bits; \overline{GAB}_H enables the outputs on the B side for the higher 16 bits; and \overline{GBA}_H enables the outputs on the A side for the higher 16 bits. These output enable signals are derived from the byte enable signals and are asserted when the slave enable signal (SEN) is activated by the M82786.

The select lines for the bidirectional transceivers allow data to flow from either the latched data or the input pins. These lines, which are not shown, can be hardwired.

The data bus controller provides the read (\overline{RD}) and write (\overline{WR}) commands, memory or I/O signal ($\overline{M}/\overline{IO}$), and a \overline{READY}_0 signal. This circuit generates two read or write commands for every 32-bit data transfer to or from the 80960MC processor (one for each 16-bit data transfer). The data bus controller starts counting clock cycles when the M82786-CS and CYCLE-IN-PROGRESS signals are asserted. At the proper time (based upon clock counts), it asserts the read/write command. The data bus controller produces \overline{READY} after receiving the SEN signal from the M82786. \overline{READY} resets the count, and another read/write command is generated.

The address translator performs four functions: it converts the four byte enable signals to A_0 , A_1 , and \overline{BHE} ; it increments A_1 after receiving \overline{READY}_0 for the first 16-bit transfer; it generates the clock signal (\overline{CBA}_L) that latches the first 16-bit data word in the bidirectional transceivers when the 80960MC processor performs a read operation; and it generates the \overline{READY} signal for the CPU.

Not shown is the cycle detector circuit that generates the CYCLE-IN-PROGRESS signal. This signal can be generated by using the circuit similar to the one shown in Figure 5-2. The start of the cycle can be detected by gating the \overline{ADS} and \overline{DEN} signals. The end of the cycle can be indicated by \overline{READY} .



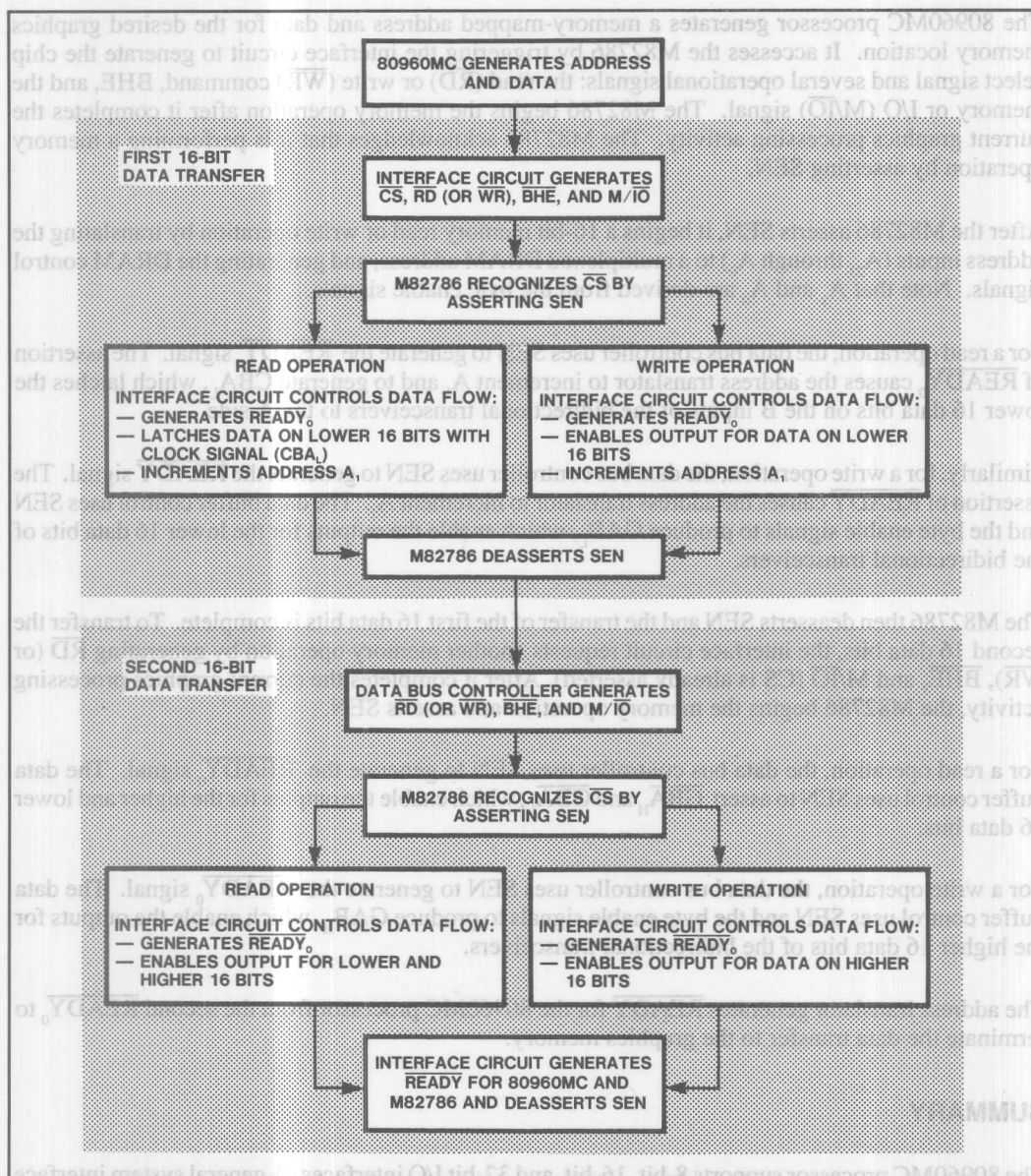


Figure 5-9: Operational Flow Diagram For M82786 Interface Circuit

Operation

The interaction between the M82786 and the 80960MC processor is summarized in Figure 5-9. The operation is divided into two 16-bit data movements for both a read and write operation.

The 80960MC processor generates a memory-mapped address and data for the desired graphics memory location. It accesses the M82786 by triggering the interface circuit to generate the chip select signal and several operational signals: the read (\overline{RD}) or write (\overline{WR}) command, BHE, and the memory or I/O ($\overline{M/\overline{IO}}$) signal. The M82786 begins the memory operation after it completes the current graphics processing activity. The M82786 acknowledges that it is performing a memory operation by asserting SEN.

After the M82786 asserts SEN, it begins a 16-bit memory read or write operation by translating the address inputs (A_{21} through A_0) to a multiplexed DRAM address, and generating the DRAM control signals. Note that A_1 and A_0 are derived from the byte enable signals.

For a read operation, the data bus controller uses SEN to generate the \overline{READY}_0 signal. The assertion of \overline{READY}_0 causes the address translator to increment A_1 and to generate CBA_L , which latches the lower 16 data bits on the B inputs of the bidirectional transceivers to the A side.

Similarly, for a write operation, the data bus controller uses SEN to generate the \overline{READY} signal. The assertion of \overline{READY} causes the address translator to increment A_1 . The data buffer control uses SEN and the byte enable signals to produce GAB_L , which enable the outputs for the lower 16 data bits of the bidirectional transceivers.

The M82786 then deasserts SEN and the transfer of the first 16 data bits is complete. To transfer the second 16 data bits, the interface circuit requests another memory operation by generating \overline{RD} (or \overline{WR}), BHE, and $\overline{M/\overline{IO}}$ (\overline{CS} is already asserted). After it completes the current graphics processing activity, the M82786 begins the memory operation and asserts SEN.

For a read operation, the data bus controller uses SEN to generate the \overline{READY}_0 signal. The data buffer control uses SEN to assert \overline{GBA}_H and \overline{GBA}_L , which enable the outputs for the higher and lower 16 data bits.

For a write operation, the data bus controller uses SEN to generate the \overline{READY}_0 signal. The data buffer control uses SEN and the byte enable signals to produce \overline{GAB}_H , which enable the outputs for the higher 16 data bits of the bidirectional transceivers.

The address translator generates \overline{READY} for the 80960MC processor from the second \overline{READY}_0 to terminate the data transfer to the graphics memory.

SUMMARY

The 80960MC processor supports 8-bit, 16-bit, and 32-bit I/O interfaces. A general system interface circuit can be designed that connects to many slave-type peripherals. This interface can be expanded to accommodate a bus master peripheral or a 32-bit to 16-bit data bus translator. These interfaces were illustrated by three design examples.

***80960MC Multiprocessor
System Architecture***

6

6

80960MC Multiprocessor System Architecture

CHAPTER 6

80960MC MULTIPROCESSOR SYSTEM ARCHITECTURE

This chapter illustrates the flexibility of the 80960MC system architecture using the advanced 32-bit 80960MC processor and the 80965 Bus Extension Unit (BXU) in a multiprocessor design. System configurations are examined from a general perspective to highlight overall the design concepts. The details of system design are discussed in subsequent chapters.

OVERVIEW OF A MULTIPROCESSOR SYSTEM ARCHITECTURE

Modules form the natural boundaries for the hardware system architecture, as shown in Figure 6-1. Each module consists of components attached to its own local bus (L-bus). The modules are interconnected by a high bandwidth 32-bit multiplexed and packetized Advanced Processor (AP) bus, which transfers data at a peak rate of 42M bytes per second at 16 MHz. The 82965 Bus Extension Unit (BXU) provides the gateway between the L-bus and AP-bus, and performs several other functions, such as supporting an optional L-bus cache.

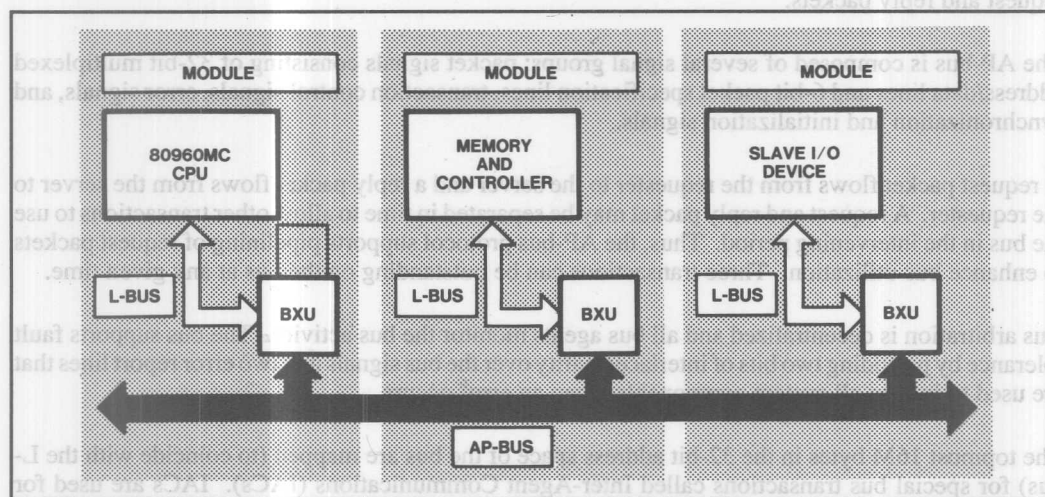


Figure 6-1: Basic 80960MC System Configuration

The L-bus

The L-bus is used to connect the components within the module, which may include processors, memory arrays, and peripherals. In a multiprocessor system, each module contains an L-bus, which is typically confined to a single board.

As described in the detail in chapter 3, the 32-bit L-bus is a high bandwidth, multiplexed bus which supports burst transactions, and can access up to four data words at a maximum rate of one word every

bus cycle. The L-bus consists of two groups of signals: address/data and control. The bus has a single fixed timing, although bus transactions can be lengthened through the use of the **READY** signal (to insert wait states).

The L-bus protocol permits both primary and secondary bus masters to coexist on the bus. The secondary bus master must obtain use of the L-bus from the bus master through the use of the **HOLDR** and **HLDAR** signals. In a multiprocessor environment, a **BXU** is always used as a master in a memory module and is often used as a slave in a processor module.

Complete details of the L-bus and bus operations are discussed in Chapter 3.

The Advanced Processor Bus

The AP-bus connects the 80960MC processor modules to system memory modules and I/O modules. The AP-bus is a synchronous, packetized, 32-bit wide bus. Synchronous refers to the fact that all components in the system, including 80960MC processors and BXUs must be driven by the same clock. It is considered a packetized bus, because read and write transactions are encoded in pairs of request and reply packets.

The AP-bus is composed of several signal groups: packet signals consisting of 32-bit multiplexed address/data lines and 6-bit packet specification lines, transaction control signals, error signals, and synchronization and initialization signals.

A request packet flows from the requester to the server and a reply packet flows from the server to the requester. A request and reply packet may be separated in time to allow other transactions to use the bus in the intervening period. Thus, the AP-bus protocol supports pipelining of request packets to enhance bus utilization. Three transactions can be outstanding on the bus at any given time.

Bus arbitration is decentralized and all bus agents monitor the bus activity. The bus supports fault tolerance by providing two bits of interlaced parity over the bus signals and two error report lines that are used to inform all system components when an error occurs.

The topmost 16M bytes in the 32-bit address space of the bus are mapped (to coincide with the L-bus) for special bus transactions called Inter-Agent Communications (IACs). IACs are used for communication between 80960MC processors and for accessing the internal registers of the BXUs.

The BXU is the only component that directly attaches to the AP-bus, and it contains all necessary bus interface logic. BXUs connect to each other in the form of a matrix to allow orderly growth in the system by the addition of AP buses or modules. A 80960MC system may have up to 32 modules (the practical limitation may be 20 modules because of electrical limitations) and four AP-buses.

A more detailed discussion of the AP-bus is contained in Chapter 7.

The BXU Component

The 80960MC processor and the 82965 BXU are the central components in the multiprocessor system architecture. The BXU interconnects the L-bus and the AP-bus and implements the following functions:

1. Translation of L-bus requests from the 80960MC processor to request packets on the AP-bus.
2. Support for cache on the L-bus
3. A prefetch function for processors
4. Support for the interface to a memory subsystem

The BXU contains seven logic blocks: the AP-Bus Interface logic, the L-Bus Interface logic, Cache Support logic, Memory Support logic, IAC Support logic, I/O Prefetch logic, and Fault Tolerance logic.

The AP-Bus Interface

The AP-bus interface of the BXU provides the AP-bus signals. The AP-bus interface performs several AP-bus functions: arbitration, pipeline monitoring, address recognition, and AP-bus signal generation. The AP-bus interface provides a set of registers that allow software to define the address ranges and modes of operation.

If the system design uses multiple AP-buses, then each module uses an individual BXU to interface to each of the AP-bus's. The AP-bus interface of each BXU provides address recognizers that can be set to a predefined range of addresses. The AP-bus address space can be interleaved over two or four BXUs to enhance system performance. Up to three accesses from the module may be pending on any pair of AP-buses.

The L-Bus Interface

The L-bus interface provides a direct interface between the BXU and the L-bus. The L-bus interface can be programmed to act as either a bus master or bus slave. The address recognizers of the L-bus interface support multiple memory address ranges. When there is more than one BXU on the L-bus, each L-bus interface coordinates the activity of the BXUs to provide efficient operation with multiple AP-buses. In this case, the address recognizers of the L-bus interface may be set up to support interleaving between multiple AP-buses. This ensures that AP-bus utilization is shared approximately equally among the AP-buses in the system.

Cache Directory and Control Logic

The L-bus cache reduces the AP-bus traffic and increases overall system performance. The AP-bus traffic is reduced because the cache effectively diverts many system memory accesses to local

memory accesses. For example, when a read access is located in the cache memory, the BXU services that request directly from the cache, without generating corresponding bus cycles on the system bus. This boosts the system's performance because the cache can provide data to the requester on the L-bus faster than requests that are serviced by a memory module.

The BXU provides the cache directory, the control logic, and the coherency logic. The coherency logic ensures that 80960MC processor uses the most recent data that is in global memory.

The data storage resides on the L-bus in external SRAM components to allow access to a large cache. This memory is configured as a two-way set associative cache.

Memory Support Logic

The BXU provides specific support facilities for memory modules. The control of a memory module is done jointly between the BXU and a memory controller. The BXU provides the signals for the AP-bus interface and access to specific registers maintained by the memory controller. The memory controller is responsible for the detailed timing, control, and direct interfacing of the memory components. Normally the memory components are DRAMS, but other memory types can be supported as well.

IAC Support Logic

IAC messages are used by 80960MC processors to communicate interrupts and other information on the AP-bus. Because the 80960MC processors are not directly attached to the AP-bus, the BXUs act as the receivers for IAC messages on behalf of the 80960MC processors in their module.

The BXU provides two registers for each of the two processors that can reside in a module. These registers store the data message and indicate the priority of the message. When an IAC message is received for one of the processors, the BXU checks the priority of the message and the status of the message data storage. Based on this information, the message is either accepted or rejected.

I/O Prefetch Logic

The I/O Prefetch logic of the BXU increases bandwidth and decreases latency for I/O accesses. Two prefetch channels handle the sequential I/O data transfers. Each channel contains two 16-byte buffers. After an I/O channel is initialized by receipt of a start command, the requests are serviced by the channel prefetch buffers.

As data is requested from one of the buffers, the BXU automatically prefetches the next data block, and stores it in the other buffer. The prefetch function provides a significant increase in I/O performance because the data requests are handled immediately from the prefetch buffers.

Fault Tolerance Support

In fault-tolerant systems with two or four AP-buses, the BXUs in a module operate in pairs as backup components for each other. For example, if an AP-bus fails, the BXU for that bus isolates itself from the failed bus, while the backup BXU services the requests directed to the failed module. This action of the BXU allows duplicate AP-buses to guard against single bus failures. To provide backup capabilities, each BXU tracks the accesses that are handled by its partner BXU. The BXU provides this fault tolerance logic, which is described in Part III of this manual.

Modes of Operation

The BXU has two modes of operation: PROCESSOR MODE or MEMORY MODE. When Processor mode is selected, the BXU can operate as a L-bus master or slave. In this mode, the BXU supports the cache, I/O prefetch, and IAC message functions. When Memory mode is selected, the BXU always operates as a bus master and generates the required L-bus signals.

A more detailed discussion of the BXU is contained in Chapter 8.

SYSTEM CONFIGURATIONS

A multiprocessor 80960MC based system is composed of a set of modules connected to AP-buses. Figure 6-2 shows three types of modules: active, passive, and the combination of an active and passive (active/passive). These three types of modules are used to build a multiprocessor system.

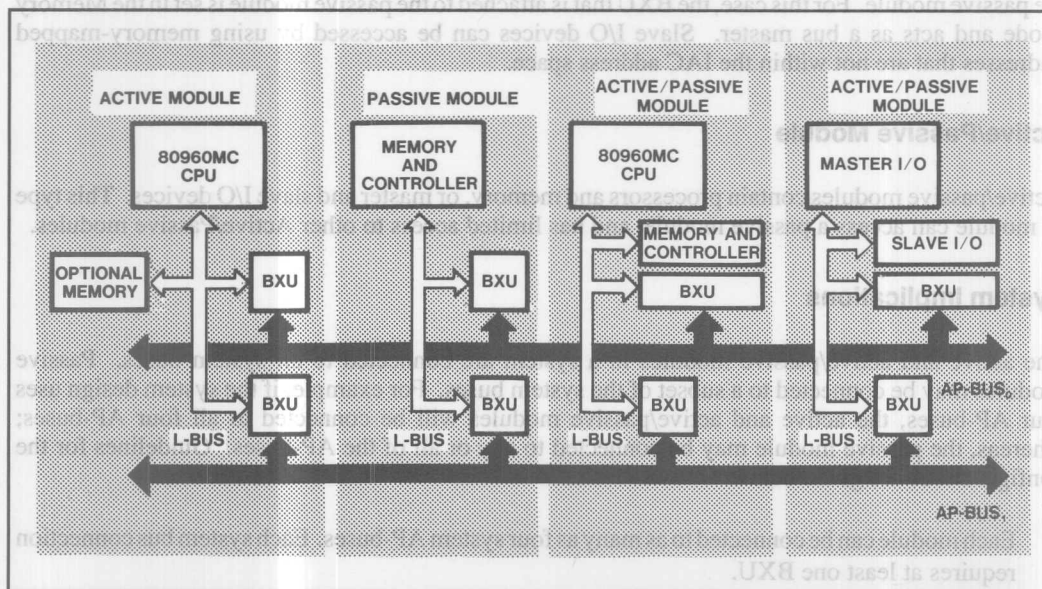


Figure 6-2: Types Of Modules

Active Module

The active module consists of at least one 80960MC processor and L-bus. Private or cache memory may be attached to the L-bus in this module. The BXU translates the L-bus signals to AP-bus signals.

In the active modules, the L-bus traffic flows either on the L-bus to its resources (cache, etc) or flows through the BXU onto the AP-bus. No request packets, however, flow from the AP-bus through the BXU to the L-bus.

For IAC transactions, the BXU acts as an extension of the processors. The BXU informs the processor of the impending IAC transaction by asserting the \overline{IAC} pin of the 80960MC processor.

In an active module, the BXU is set to the Processor Mode. When operating in this mode, the BXU does not need control of the bus, and consequently, does not need to arbitrate for control of the L-bus.

Passive Module

The passive module consists of memory or a slave I/O device, the BXU, and the L-bus. The L-bus connects the BXU to the memory or I/O device. The BXU translates AP-bus signals to L-bus signals.

In the passive modules, all L-bus requests originate from the source agent (generally, the 80960MC processor) of an active module via the AP-bus. Bus traffic flows from the source agent through a BXU attached to the active module, onto the AP-bus, and finally through another BXU attached to the passive module. For this case, the BXU that is attached to the passive module is set in the Memory Mode and acts as a bus master. Slave I/O devices can be accessed by using memory-mapped addresses that are not within the IAC address space.

Active/Passive Module

Active/passive modules contain processors and memory, or master and slave I/O devices. This type of module can access a passive module, and has limited access to other Active/Passive modules.

System Implications

The active and active/passive modules in a system are connected to all system buses. Passive modules may be connected to a subset of the system buses. For example, if the system design uses four AP-buses, the active and active/passive modules will be connected to all four AP-buses; whereas, the passive module may be connected to any or all of the AP-buses. Guidelines for the configurations are given below:

- Each module can be connected to as many as four system AP-buses. Each system bus connection requires at least one BXU.

- Each module can support up to two 80960MC processors (limited by the BXU's support for IAC messages). The number of other components is only limited by the electrical and physical constraints of the module implementation.
- Logical addressing allows up to 32 modules for every system, although electrical considerations may limit the practical number of modules to 20.

SUMMARY

The basic hardware system configuration has been designed to be both modular and flexible. It is comprised of active, passive, and active/passive modules that form natural system boundaries. The high-bandwidth AP-bus is used for the data path between the modules. Each module contains a BXU, which interfaces directly to the AP-bus and L-bus. To accomplish this task, the BXU contains seven logic blocks.

This chapter presented an overview for basic hardware system design. The next three chapters discuss the details of the AP-bus, the BXU, and the memory and I/O interface.

CHAPTER 7 ADVANCED PROCESSOR BUS

Efficient bus utilization is essential in a multiprocessing system. A simple and efficient approach to building an 80960MC processor interconnect system is to use the Advanced Processor bus (AP-bus). The AP-bus protocol divides bus transactions into request and reply packets. Up to three request packets can be outstanding on the AP-bus before a reply packet must be received. In this way, the AP-bus reduces bus occupancy and increases the performance of 80960MC multiprocessor systems.

This chapter describes the AP-bus operation and covers the topics shown below.

- An overview of the AP-bus topology and description of the AP-bus signals
- Memory organization and AP-bus transactions including memory and IAC transactions
- AP-bus protocol and signal timing

AP-BUS OVERVIEW

The AP-bus forms the data communication path between the system modules. Access to the AP-bus is made possible by the BXU. The 80960MC processor utilizes the AP-bus to fetch instructions and to manipulate information from both memory and I/O devices. The AP-bus has the following features:

- 32-bit multiplexed address/data path
- High data bandwidth
- Four-word burst capability
- High bus utilization by supporting up to three outstanding requests at one time with intermixed requests and replies.
- Transparent arbitration that allows the addition/removal of bus agents without modifying any hardware.
- Interlaced parity

The AP-bus is a synchronous packet bus that consists of several signal groups: packet signals that include 32 multiplexed address/data lines and six packet specification lines, transaction control signals, error signals, and synchronization and initialization signals. A transaction on the bus is separated into a request packet that flows from the requester to the server and a reply packet that flows from the server to the requester.

A request and reply may be separated in time to allow other transactions to use the bus in the intervening period. This provides a pipeline feature that enhances bus utilization. Bus arbitration is decentralized and all bus agents monitor the bus activity. The AP-bus supports fault tolerance by providing interlaced parity over the address and packet specification lines, signal duplication on the transaction control lines, and a bus timer used to monitor the AP-bus for non-response to an

outstanding request. Redundant error report lines are provided to inform all system components when an error occurs.

AP-Bus Topology

Figure 7-1 shows a typical system configuration that uses two AP-buses (up to four AP-buses are allowed). The BXU provides the interface between the L-bus and the AP-bus. The BXU contains several programmable registers, which control its operation. See Chapter 8 for a detailed description of the BXU programmable register array.

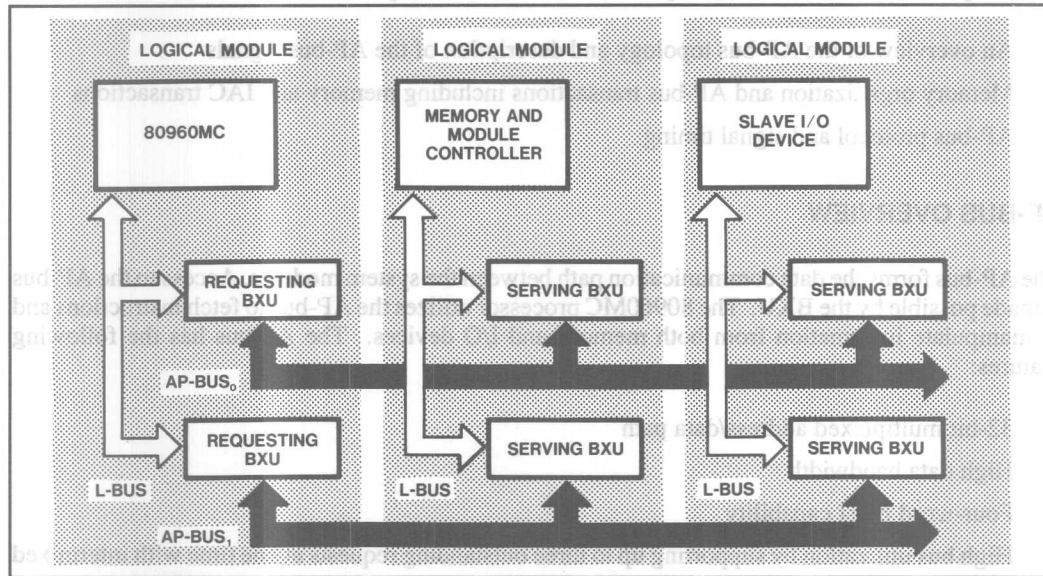


Figure 7-1: AP-Bus Topology

Each BXU (bus agent) has three identification values: logical, physical, and bus. All the bus agents within the same logical module have the same logical identification value, which is assigned by the system software. This value is stored in the *Logical-ID* register of the BXU (see Appendix A for the description of the *Logical-ID* register). Each BXU on a particular bus has a unique physical identification assigned during initialization. Each bus has a unique identification value (zero through three).

There are two types of agents: requesting and serving. The requesting agents are the BXUs attached to the active or active/passive modules. These BXUs translate the 80960MC processor signals to the AP-bus signals. The serving BXU receives the AP-bus signals, performs the desired functions, and passes the data back to the requesting 80960MC processor. Bus agents do not initiate any AP-bus operations; they simply carry out the operations specified by the 80960MC processor.

AP-BUS SIGNAL GROUPS

Figure 7-2 shows the four AP-bus signal groups: packet (address/data and specification lines), transaction control, error, and synchronization and initialization. This section presents general definitions of the AP-bus signal groups. Complete details of the signal descriptions and relationships are provided in subsequent sections.

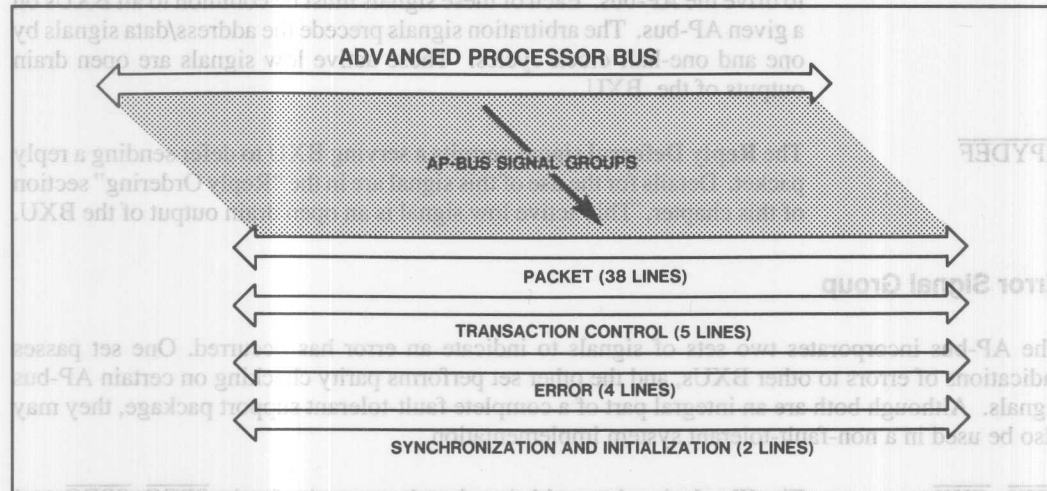


Figure 7-2: AP-Bus Signal Groups

Packet Signal Group

The Packet signal group consists of 38 bidirectional lines that transmit the address, data, and transaction type. The address and data signals are multiplexed.

$\overline{\text{SPEC}}_5$ - $\overline{\text{SPEC}}_0$

The **Specification** signals define the packet type and other parameters required for the bus transaction. Details of these signals are described in the "Bus Transaction" section of this chapter. These active low signals are open drain outputs of the BXU.

$\overline{\text{AD}}_{31}$ - $\overline{\text{AD}}_0$

The system **Address/Data**₃₁ through system **Address/Data**₀ represent the **address** signals on the AP-bus during the address cycle and **data** signals on the AP-Bus during **data** cycles. $\overline{\text{AD}}_{31}$ is the most significant bit and $\overline{\text{AD}}_0$ is the least significant bit. These active low signals are open drain outputs of the BXU.

Transaction Control Signal Group

The Transaction control group consists of 5 bidirectional signals that control the actual sequencing of packets on the AP-bus. They consist of four arbitration signals (\overline{ARB}_3 - \overline{ARB}_0) and a signal that can defer a reply packet (RPYDEF).

\overline{ARB}_3 - \overline{ARB}_0

The four **Arbitration** signals determine which agent gains exclusive access to drive the AP-bus. Each of these signals must be common to all BXUs on a given AP-bus. The arbitration signals precede the address/data signals by one and one-half clock cycles. These active low signals are open drain outputs of the BXU.

\overline{RPYDEF}

The **Reply Deferral** signal permits a serving BXU to defer sending a reply packet. Details for the use of this signal are in the "Reply Ordering" section of this chapter. This active low signal is an open drain output of the BXU.

Error Signal Group

The AP-bus incorporates two sets of signals to indicate an error has occurred. One set passes indications of errors to other BXUs, and the other set performs parity checking on certain AP-bus signals. Although both are an integral part of a complete fault-tolerant support package, they may also be used in a non-fault-tolerant system implementation.

\overline{CHK}_1 - \overline{CHK}_0

The **Check** signals provide interlaced even parity for the \overline{SPEC}_5 - \overline{SPEC}_0 and \overline{AD}_{31} - \overline{AD}_0 lines. These active low signals are open drain outputs of the BXU.

\overline{BERL}_1 - \overline{BERL}_0

The **Bus Error** report line signals indicate that an error has been detected in the system. These active low signals are open drain outputs of the BXU.

Synchronization and Initialization Signal Group

The synchronization signal (CLK2) provides the capability for all AP-bus agents to operate with the proper timing relative to each other. The initialization signal (RESET) brings all AP-bus agents to a consistent state.

CLK2

The **System clock** provides the fundamental timing and synchronization for all transactions performed by the AP-bus agents. The clock rate is twice the frequency of a bus cycle. For example, if the bus cycle frequency is 16 MHz, then the CLK2 frequency is 32 MHz.

RESET

The assertion of the **RESET** signal forces all AP-bus agents to reset and synchronize. After the first system clock period begins, all bus agents remain in synchronization. The assertion of the RESET signal is the only method available to synchronize all the bus agents for proper operation.

Bus Signal Summary

Table 7-1 shows the summary of the AP-bus signals.

Table 7-1: AP-Bus Signal Summary

Signal Group	Symbol	Function
Packet	$\overline{\text{SPEC}}_5\text{-}\overline{\text{SPEC}}_0$	Specify the type of packet
	$\overline{\text{AD}}_{31}\text{-}\overline{\text{AD}}_0$	32-bit address
	$\overline{\text{AD}}_{31}\text{-}\overline{\text{AD}}_0$	32-bit data
Transaction Control	$\overline{\text{ARB}}_3\text{-}\overline{\text{ARB}}_0$	Arbitrate for AP-bus access
	$\overline{\text{REPDEF}}$	Defers reply packet
Error	$\overline{\text{CHK}}_1\text{-}\overline{\text{CHK}}_0$	Provide parity checks
	$\overline{\text{BERL}}_1\text{-}\overline{\text{BERL}}_0$	Indicate AP-bus error
Synchronization and Initialization	CLK2	Clock signal (double the bus frequency)
	RESET	Resets bus agent to a known state

NON AP-BUS MODULE SUPPORT SIGNALS

These signals are point-to-point connections and are not considered part of the AP-bus. The $\overline{\text{INITID}}$ and $\overline{\text{COM}}$ signals are valid during initialization, and the $\overline{\text{MODCHK}}$, $\overline{\text{BOUT}}$, $\overline{\text{COM}}$, and $\overline{\text{V}}_{\text{REF}}$ signals are required for fault tolerant designs to detect the source of an error. $\overline{\text{POPQUE}}$ and $\overline{\text{SSBUSY}}$ are used to co-ordinate activities between AP-Buses within a system.

$\overline{\text{INITID}}$

The **Initialization Identification** signal provides a way to access a single BXU during initialization. The $\overline{\text{INITID}}$ pin of the BXU is physically connected to one of the 32 AP-Bus address/data lines. The $\overline{\text{INITID}}$ signal together with "Identify Device Order" IAC provide a unique address for the bus agent during initialization.

$\overline{\text{MODCHK}}$

The **Module Check** signal is used in fault-tolerant system design and is described in detail in Chapter 11.

$\overline{\text{BOUT}}$

The **Bus Output Control**, when asserted, indicates that a bus agent is driving the AP-bus. This signal is used in fault-tolerant system design and is described in detail in Chapter 11.

COM

The Communication signal is used to load information into a BXU as part of the initialization sequence or to inform external logic that the component has failed. This signal is not involved in any aspect of the AP-bus operation, but is provided to simplify loading board-dependent information into the BXU. The COM signal is also used to indicate external errors in fault-tolerant system designs. Complete details of this signal are provided in the “Serial COM Protocol” section of Chapter 14.

V_{REF}

The Voltage Reference pin provides a stable voltage reference for the AP-bus input buffers of the BXU. External hardware must provide a nominal 2.0V on the VREF pin (see the M82965 BXU data sheet for the exact specification) during normal operation. The VREF pin is used to distinguish between a cold RESET (memory of errors cleared) and a warm RESET (memory of errors retained). Complete details about this signal are provided in Chapter 14.

POPQUE

The **Pop Queue** signal is used in fault tolerant system design and is described in Chapter 13.

SSBUSY

The **Subsystem Busy** signal is used to coordinate activity between AP-buses within a system. It is described in Chapter 13.

MEMORY ORGANIZATION

The AP-bus provides a four gigabyte address range to access memory and memory-mapped devices. The address range is accessible in four word blocks (each 32 bit word contains four 8-bit bytes). By organizing memory in this manner, the entire 4 Gbyte address range is logically accessible on word boundaries starting at address 0000 0000_H. The memory address (with the 4 least significant bits = 0) present on the AP-Bus during the address cycle points to the first word within the block boundary. The source BXU can specify individual bytes within a word during a write transaction data cycle by encoding “BYTE MARKS” on the AP-Bus **SPEC** lines.

A single AP-bus transaction can access a maximum of four words and cannot cross the boundary of a block. To achieve the lowest number of memory cycles for a transaction, the following rules must be adhered to: single word accesses must be word aligned (least two significant bits = 0). A transaction may read or write a contiguous string of one to sixteen bytes of data within the block. The 80960MC processor will break any program's request of more than 16 bytes or accesses that would have necessitated crossing a block boundary into multiple transactions.

The uppermost 16 Mbyte addresses of the 4 Gbyte address space are reserved for IACs; direct communication between bus agents (see the “IAC Transaction” section in this chapter). This specific set of memory-mapped addresses is recognized by all AP-bus agents to facilitate interagent message communications. IACs are used for system functions, such as initialization, access to registers in the AP-bus agents, and interrupt handling.

BUS TRANSACTIONS

AP-bus agents communicate with each other by exchanging packets of information. A packet is a sequential group of bus cycles that contains information on the type of operation, the address location, and the number of data bytes to transfer. Two packets, a **request** and **reply**, form an AP-bus transaction.

A request packet initiates an AP-bus transaction, and the reply packet completes the transaction. The request packet specifies three items: the type of operation, the location of the requested device, and the number of data bytes. The request packet includes data if a write operation is performed. The reply packet responds to the request packet by indicating the status of the action requested. For a read operation, the reply packet also responds with the requested data.

Figure 7-3 shows the different types of request packets. The request packets are divided into two basic groups of operations: a read group that transfers data to the 80960MC processor, and a write group that transfers data from the 80960MC processor. Each group contains specific operations, such as read word(s) or Read Modify Write (RMW) operations. The read byte or read double-byte operations are optimized for memory-mapped I/O devices to facilitate the interface to an 8-bit or 16-bit bus. The RMW operations are used to perform "Atomic Accesses". In an Atomic Access the BXU guarantees that once a processor begins a read-modify-write operation on a set of memory locations, it is allowed to complete the operation before another processor is allowed to access the same location. The specification lines are used to indicate which function is performed. Complete details of the specific operations are described in this section.

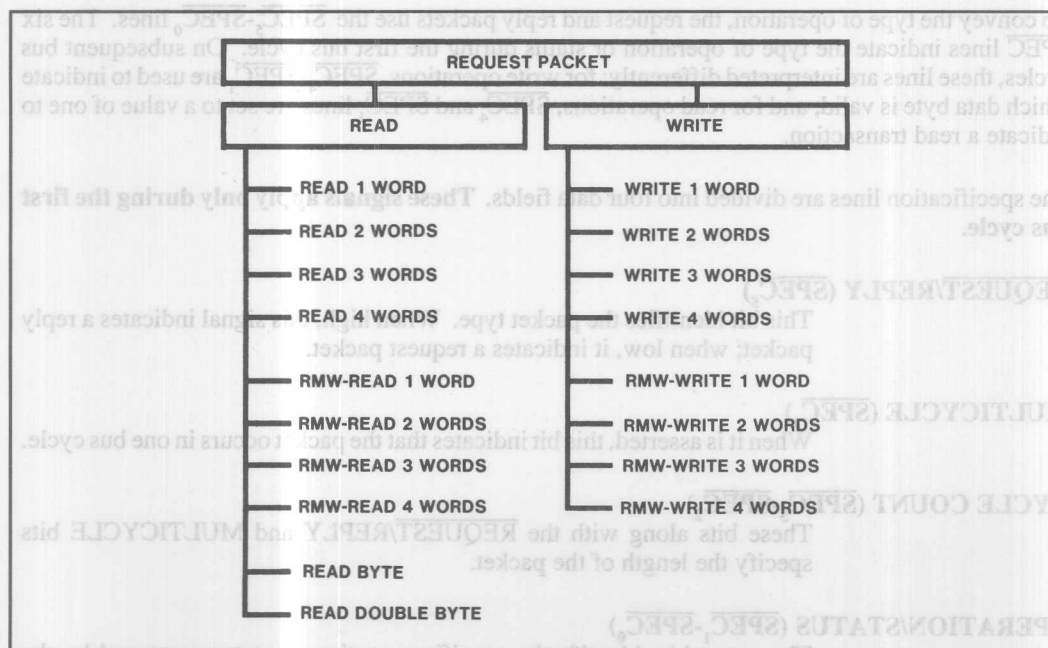


Figure 7-3: Request Packet Organization

The reply packets are also divided into two basic status groups that indicate acceptance or refusal of the requested operation, as listed in Figure 7-4. Each group contains specific responses, which are discussed in this section.

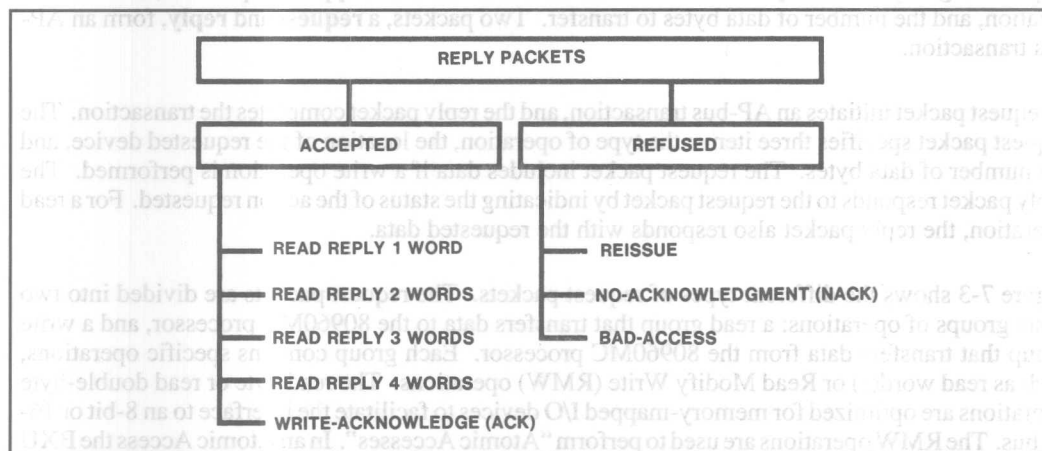


Figure 7-4: Reply Packet Organization

Transaction Specification

To convey the type of operation, the request and reply packets use the $\overline{\text{SPEC}}_5$ - $\overline{\text{SPEC}}_0$ lines. The six $\overline{\text{SPEC}}$ lines indicate the type of operation or status during the first bus cycle. On subsequent bus cycles, these lines are interpreted differently: for write operations, $\overline{\text{SPEC}}_4$ - $\overline{\text{SPEC}}_1$ are used to indicate which data byte is valid; and for read operations, $\overline{\text{SPEC}}_4$ and $\overline{\text{SPEC}}_1$ lines are set to a value of one to indicate a read transaction.

The specification lines are divided into four data fields. These signals apply only during the first bus cycle.

REQUEST/REPLY ($\overline{\text{SPEC}}_5$)

This bit identifies the packet type. When high, this signal indicates a reply packet; when low, it indicates a request packet.

MULTICYCLE ($\overline{\text{SPEC}}_4$)

When it is asserted, this bit indicates that the packet occurs in one bus cycle.

CYCLE COUNT ($\overline{\text{SPEC}}_3$ - $\overline{\text{SPEC}}_2$)

These bits along with the REQUEST/REPLY and MULTICYCLE bits specify the length of the packet.

OPERATION/STATUS ($\overline{\text{SPEC}}_1$ - $\overline{\text{SPEC}}_0$)

These two bits identify the specific operation or status conveyed by the packet.

Table 7-2 shows the relationship between the encoded $\overline{\text{SPEC}}$ lines and the specific operation of the request or reply packet during the first bus cycle. Note that the encoded $\overline{\text{SPEC}}$ lines do not use all the possible combinations. The values that are not encoded are unused and reserved for future product enhancements. Table 7-2 also shows the number of bus cycles and the number of words requested or transferred for a particular packet.

Table 7-2: Specification Encodings for Packets

Packet	Category	Function	Specification Lines (SPEC ₅ -SPEC ₀)						Hexi-Decimal Equiv.	Bus Cycles	Number of Words Requested (Transferred)
			Binary								
			5	4	3	2	1	0			
Request	Read	Read 1 word	1	0	0	0	0	0	20	1	1
		Read 2 words	1	0	0	1	0	0	24	1	2
		Read 3 words	1	0	1	0	0	0	28	1	3
		Read 4 words	1	0	1	1	0	0	2C	1	4
		RMW-Read 1 word	1	0	0	0	0	1	21	1	1
		RMW-Read 2 words	1	0	0	1	0	1	25	1	2
		RMW-Read 3 words	1	0	1	0	0	1	29	1	3
		RMW-Read 4 words	1	0	1	1	0	1	2D	1	4
		Read byte	1	0	0	0	1	0	22	1	0.25
	Read double byte	1	0	0	0	1	1	23	1	0.5	
	Write	Write 1 word	1	1	0	0	0	0	30	2	(1)
		Write 2 words	1	1	0	1	0	0	34	3	(2)
		Write 3 words	1	1	1	0	0	0	38	4	(3)
		Write 4 words	1	1	1	1	0	0	3C	5	(4)
		RMW-Write 1 word	1	1	0	0	0	1	31	2	(1)
		RMW-Write 2 words	1	1	0	1	0	1	35	3	(2)
		RMW-Write 3 words	1	1	1	0	0	1	39	4	(3)
		RMW-Write 4 words	1	1	1	1	0	1	3D	5	(4)

Table 7-2: Specification Encodings for Packets (cont.)

Packet	Category	Function	Specification Lines (SPEC ₅ -SPEC ₀)						Hexi-Decimal Equiv.	Bus Cycles	Number of Words Requested (Transferred)
			Binary								
			5	4	3	2	1	0			
Reply	Accepted	Read reply 1 word	0	0	0	1	1	0	06	1	(1)
		Read reply 2 words	0	1	0	0	1	0	12	2	(2)
		Read reply 3 words	0	1	0	1	1	0	16	3	(3)
		Read reply 4 words	0	1	1	0	1	0	1A	4	(4)
		Write-Acknowledge	0	0	0	0	1	0	02	1	N/A
	Refused	Reissue	0	0	1	1	1	1	0F	1	N/A
		No-Acknowledgment	0	0	1	0	1	0	0A	1	N/A
		Bad-access	0	0	1	0	1	1	0B	1	N/A

NOTE: Binary 1 indicates that the referenced SPEC line is asserted (i.e., driven low).

Note: Binary 1 indicates that the referenced SPEC line is asserted (i.e., driven low).

After the first bus cycle, the SPEC lines represent other information for the write and read operations. For a write operation, four of the SPEC lines are used to specify which data bytes to write during the current bus cycle (see the “Write Data Transfer” section of this chapter for more details). For a read operation, three SPEC lines are used to indicate that data is being transferred (see the “Read Data Transfer” section for more details).

Request Packets and Accepted Replies

Data is transferred on the AP-bus by request and reply packets. A write-request packet transfers data from the requesting agent to the serving agent. A read-reply packet transfers data from the serving agent to the requesting agent.

A word (32-bits) is the basic unit of measure for a data transfer, although individual data bytes (8 bits) can be transferred in a write request packet. Individual data bytes are transferred during a write operation by using four SPEC lines after the first bus cycle as byte marks. For a read operation, the entire data word is transferred and the 80960MC processor extracts the desired data bytes.

The write-request packets or the read-reply packets transfer up to four words of data on the \overline{AD} lines in two to five bus cycles depending on the amount of data to transfer (a four-word write-request is five bus cycles long - one bus cycle to transmit the address and four bus cycles to transmit the data). For multiple word data accesses that do not start at word boundaries (i.e., Word₀-Byte₁), the BXU may require an additional cycle to properly align the data bytes before transferring them onto the AP-bus. The following examples of read and write packet transactions contained in this section illustrate details of these functions.

Read Data Transfer

The following sequence of events occurs during a read operation:

- An 80960MC processor initiates a read operation on the L-bus by issuing a read command, supplying an address, and designating how many words to read through the SIZE signals.
- The requesting bus agent (BXU) translates these signals into a read-request packet on the AP-bus.
- The serving bus agent retrieves the data from the memory or I/O device and sends a read-reply packet to the requesting bus agent.
- The requesting bus agent receives the data and transfers it to the L-bus, thus completing the cycle.

The read transaction transfers one to four words of data. The address in the read-request packet points to the first data word to be read. Read transactions transfer a single word (Byte₃, Byte₂, Byte₁, and Byte₀) in one bus cycle. Additional words require additional bus cycles.

Individual bytes can be read, but only if the 80960MC processor extracts the desired data bytes and ignores the others. The BXU does not have the capability to fragment a word, and thus will always place a whole word on the L-bus. To read a string of data bytes that cross a word boundary within a block requires additional bus cycles, even if only two data bytes are requested. For example, if the desired data string is Word₀-Byte₃ and Word₁-Byte₀, two bus cycles are required because two words are read: one to read Word₀ and the other to read Word₁. In this case, the 80960MC receives both words and extracts the bytes desired.

The following example depicts the data movement for a typical read transaction. Assume that an 80960MC processor needs to read three words from a single block in memory located at 0000 0040_H. Table 7-3 shows the memory block organization with the letters A through P representing 16 individual data bytes of the four words. This transaction is comprised of a one cycle request packet and a 3-cycle reply packet.

The requesting bus agent places a **request packet** on the AP-bus, as shown in Table 7-4. The single-cycle request packet specifies that a three-word read operation is requested at location 0000 0040_H. The serving bus agent uses the read-request packet to read three words from the specified block of memory. The four byte word is the basic unit of transfer.

Table 7-3: Memory Block Data for Read Example

	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Word ₃	P	O	N	M
Word ₂	L	K	J	I
Word ₁	H	G	F	E
Word ₀	D	C	B	A*

NOTE: * Address 0000 0040_H points to this location.
Letters represent one data byte.

Table 7-4: Read-Request Packet

	Specification Lines						Address/Data Lines			
	5	4	3	2	1	0	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Bus Cycle ₁	1	0	1	0	0	0	00 _H	00 _H	00 _H	40 _H

The transaction is completed by a three-cycle **read-reply** packet that contains the requested information. Table 7-5 shows the alignment of the data bytes with respect to the \overline{AD} lines in the reply packet. Because three words are read, the transaction requires three bus cycles: the first bus cycle contains the data word of the original address, the second and third bus cycles follow with the second and third data words. During the first bus cycle \overline{SPEC}_3 - \overline{SPEC}_0 contain the binary code 010110, indicating a "READ REPLY 3 WORD PACKET". During the second and third bus cycles, \overline{SPEC}_5 and \overline{SPEC}_0 remain deasserted, \overline{SPEC}_4 and \overline{SPEC}_1 remain asserted and \overline{SPEC}_3 and \overline{SPEC}_2 are "don't cares".

Table 7-5: Read-Reply Packet

	Specification Lines						Address/Data			
	5	4	3	2	1	0	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Bus Cycle ₁	0	1	0	1	1	0	D	C	B	A
Bus Cycle ₂	0	1	x	x	1	x	H	G	F	E
Bus Cycle ₃	0	1	x	x	1	x	L	K	J	I

NOTES:

1. Capital letters represent one data byte.
2. "x" means "don't care."

Read Byte and Read Double-Byte

The read byte or read double-byte are special request packets that can facilitate memory-mapped operations to peripheral devices on an 8-bit or 16-bit bus attached to a BXU on an I/O module. By specifying whether an access to a memory-mapped device is a byte or double-byte, the serving bus agent on the AP-bus can formulate a proper request its L-Bus.

The read-byte request accesses a byte in a single cycle by using the two low-order address bits to point to the requested data byte. The data byte is transferred in its normal location on the \overline{AD} lines of the AP-bus. The read double-byte accesses any two-byte string within a block of address space. If the address points at the last byte of a word, then two bus cycles are required to transfer the two bytes as they overlap a word boundary.

Both of these special packets are useful when the serving bus agent interfaces to either an 8-bit or 16-bit local bus. For normal memory accesses (i.e., for non-memory-mapped accesses), these request packets operate like the read-request packet.

Write Data Transfer

The following sequence of events that occur during a write operation, are similar to a read operation, except that the data flows from the source agent.

- An 80960MC processor initiates a write operation on its L-bus by issuing a write command, supplying an address, and designating how many words to write.
- The requesting bus agent (BXU) translates these signals into a write-request packet on the AP-bus and sends the data to the serving bus agent.
- The serving bus agent receives the data from the requesting bus agent and sends commands on its L-bus attached to write to the appropriate memory or I/O location.
- The serving bus agent sends a reply packet thus completing the cycle.

The write transaction transfers one to four words of data, with the capability to write to individual data bytes. The write-request packet transfers a single word in two bus cycles, a double word in three bus cycles, etc. The first cycle transmits the operation and address and, subsequent cycles transmit the data and the byte identification. To write a string of data bytes that cross a word boundary within a block requires additional bus cycles; one for each word boundary crossed.

Because the write transaction modifies individual data bytes, the write-request packet must include information on which bytes to alter. After the first bus cycle, the packet contains this byte information on four \overline{SPEC} lines (\overline{SPEC}_4 - \overline{SPEC}_1), which are **byte mark** signals during this time. Each byte mark corresponds to a data byte: byte Mark₀ (\overline{SPEC}_1) represents Byte₀, byte Mark₁ ($\overline{SPEC}_2) represents Byte₁, byte Mark₂ (\overline{SPEC}_3) represents Byte₂, and byte Mark₃ (\overline{SPEC}_4) represents Byte₃. The desired data byte is written by asserting the appropriate byte mark. After the first bus cycle, \overline{SPEC}_5 remains asserted to indicate that data is transferred for this write request.$

The following example depicts the data movement for a typical write transaction. In this example, the 80960MC processor writes to six bytes starting at memory location 0000 0041_H, which is located in memory block 0000 0040_H. Table 7-6 shows the memory block organization with the letters A through P representing 16 individual data bytes. This transaction comprises of a three cycle write packet and a one cycle reply (ack or nack) packet.

Table 7-6: Memory Block Data Before Write Operation

	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Word ₃	P	O	N	M
Word ₂	L	K	J	I
Word ₁	H	G	F	E
Word ₀	D	C	B	A

NOTE: Letters represent one data byte.

The three-cycle write-request packet is generated by the requesting bus agent. Table 7-7 illustrates the "WRITE-2-WORDS" command, the desired address, and the alignment of data byte marks and data on the AP-bus. The first cycle defines the specific operation and transmits the word aligned address. During the second bus cycle, the BXU transfers the data bytes of Word₀ (U, V, and W), which are designated by the byte marks. During the third bus cycle, the designated data bytes of Word₁ (X, Y, and Z) are transferred.

Table 7-7: Write-Request Packet

	Specification Lines						Address/Data			
	5	4	3	2	1	0	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Bus Cycle ₁	1	1	0	1	0	0	00 _H	00 _H	00 _H	40 _H
Bus Cycle ₂	1	1	1	1	0	x	W	V	U	—
Bus Cycle ₃	1	0	1	1	1	x	—	Z	Y	X

NOTES:

1. Capital letters represent one data byte.
2. "x" means "don't care."
3. "—" means no change to current value.

The serving agent sends a write acknowledge reply packet that signifies completion of the transaction as shown in Table 7-8. The updated memory block organization is shown in Table 7-9.

Table 7-8: Write-Acknowledge Packet

	Specification Lines						Address/Data			
	5	4	3	2	1	0	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Bus Cycle ₁	0	0	0	0	1	0	Undefined			

Table 7-9: Memory Block Data After Write

	Byte ₃	Byte ₂	Byte ₁	Byte ₀
Word ₃	P	O	N	M
Word ₂	L	K	J	I
Word ₁	H	Z	Y	X
Word ₀	W	V	U	A

NOTE: Letters represent one data byte.

Read-Modify-Write

The Read-Modify-Write (RMW) functions allow the AP-bus agents to read and modify data at a given location as a single indivisible action. The AP-bus protocol defines a RMW-READ packet to indicate the start of an indivisible operation and a RMW-WRITE packet to denote the termination of this action.

In the memory mode, the BXU provides two to four RMW locks with timeouts. Four locks are available if the module is not interleaved with other modules, and two locks if it is interleaved. The serving BXU in a memory module locks one quarter of its available memory (up to 1 Gbyte), whenever a RMW-READ is accepted into the module and AP address bits 6 and 4 match its address segment. Any other RMW-READ packets addressing this segment are rejected. This Block of memory is available, however, for any other type of memory operation by any bus. For example, the source agent may make additional read or write memory accesses, but not RMW accesses.

The RMW-READ packet can be answered with one of two reply packets: the READ-REPLY or REISSUE-REPLY. The read-reply packet returns the requested data and indicates that the lock for the appropriate Block was set. The reissue-reply packet indicates that the lock is already set and that the requesting agent must attempt the operation later. No data is returned with a reissue-reply packet.

The RMW-WRITE packet writes data and removes the lock. This packet is equivalent to a normal write-request packet except that it resets the lock for the selected memory location. The write-acknowledge reply packet indicates that the lock has been removed.

Four independent lock timers are provided in the BXU to ensure a malfunctioning agent cannot initiate a lock and deny other agents access to the system memory. The timer for a particular segment starts when its lock is asserted, and should never timeout in normal system operations. If however, a lock timeout occurs, its associated lock will be removed. The duration of the timeout is between 4096 and 8192 clock cycles.

When using the RMW operations, certain conventions must be observed by the bus agents.

1. The purpose of a RMW operation is to construct indivisible transactions. These operations should not be used to directly lock a memory data structure. The lock must be restricted to a short duration operation. In general, this means that an agent performing an indivisible sequence must guarantee that it does not suspend operation or handle an interrupt without first terminating the indivisible sequence.
2. All agents sharing a data structure or communication signal must use the same location as the starting point for their RMW operators. If the locations are not the same, the conflicting operations are not correctly sequenced (blocked).
3. Any single agent may have only a single RMW operation outstanding at any given time.

Refused Reply Packets

Besides the various reply packets that accept a request, there are three reply packets that refuse a request packet. They are the reissue, no-acknowledgement, and bad-access replies. Each of these reply packets elicit different responses from the requesting bus agent.

The REISSUE reply packet, which was briefly described in the "Read-Modify-Write" section, indicates that the serving agent was unable to reply at the prescribed time because of temporary conditions. There is no problem that prevents a successful access if the request packet is sent at a later time. The reissue reply packet is transmitted in response to a RMW-READ packet when memory is already locked, or for a request packet directed to a memory-mapped I/O device where the attached I/O bus is temporarily blocked.

The NO-ACKNOWLEDGEMENT reply packet (NACK) indicates that the request packet cannot be completed. The requesting agent must determine how to respond to this condition. NACK reply packets are used exclusively for IAC messages. For example, the NACK reply packet is issued in response to an IAC request if the serving agent has insufficient buffer space, or if the request packet does not have high enough priority. Complete details are discussed in the following "IAC transaction" section.

The BAD-ACCESS reply causes the requesting agent to terminate a transaction. One possible cause of a BAD-ACCESS simply is the transaction exceeded the bus timeout period. The requesting agent terminates its own transaction with a bad-access reply to remove the request packet from the AP-bus. The bad-access reply can be used for a variety of situations to indicate problems with the request packet. When the bad-access reply is encountered, the source agent should not try the access again.

The BAD-ACCESS reply signifies a failed access and is a valid reply to any request packet. The accessed address should be considered unavailable. The source agent must determine the appropriate recovery from this condition.

IAC TRANSACTIONS

An Interagent Communication (IAC) is a mechanism to facilitate communication between the 80960MC processors. All bus agents recognize a specific set of memory-mapped addresses (the uppermost 16M of the 4G-byte address range) as an IAC transaction.

IAC requests are split into two major groups: **message** and **register** requests. Message IACs provide a method for one processor to communicate with another processor. The register-request IACs allow direct communication to the registers in the BXU.

The information represented within a specific IAC address is: an FF_H in the uppermost eight bits (high order byte) are an IAC identification (function); the remaining 24 bits define the type of IAC, the Module Destination, L-Bus Destination, and the Internal Destination in the BXU.

IAC Flow

The 80960MC processor originates the IAC transaction by writing to a reserved memory-mapped address (top 16 Mbyte) on the L-bus. The BXU that is attached to the 80960MC processor's L-bus recognizes that address as an IAC request, and responds by performing the requested action itself or by passing the IAC request to another BXU via the message passing AP-bus. IAC's always originate on an L-Bus. If the IAC is transmitted on the AP-bus, the IAC request flows from an L-bus, onto the AP-bus, and then to another L-bus, which is its final destination.

The 80960MC processor can be connected to a maximum of four AP-buses through four BXUs, but only the AP-bus designated as the message bus during initialization transmits the IAC message. The BXUs that coexist on both the L-bus and the message AP-bus handle the IAC transactions generated by the 80960MC processors.

IAC Address Formats

The IAC is defined by a 32-bit memory-mapped address, which contains the following five address fields: *IAC identification*, *module destination*, *L-bus destination*, *IAC type*, and *internal destination*, as shown in Figure 7-5. The *IAC identification* field provides a way to designate the memory-mapped address as an IAC transaction. When this field is equal to FF_H , an IAC transaction occurs. The *module destination* field specifies which specific BXU residing on the AP-bus responds to the IAC request. The *L-bus destination* field specifically identifies which of four possible BXU's residing on the processors local bus will either act on the request or propagate it onto the AP-Bus. The *IAC type* field specifies the type of IAC transaction. The multipurpose *internal destination* field specifies various commands, BXU register addresses, or IAC message priorities.

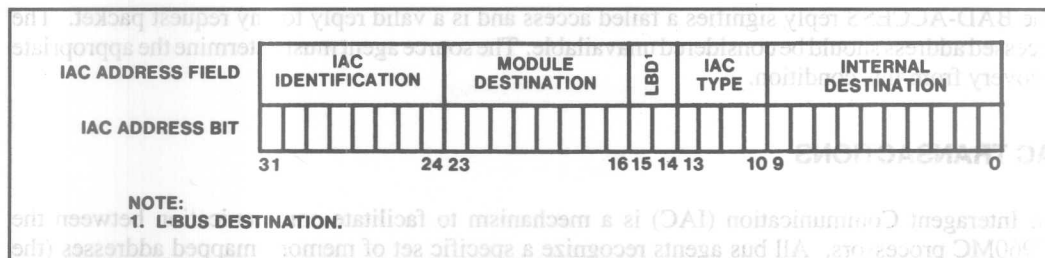


Figure 7-5: IAC Address Format

The 80960MC processor can initiate the six types of IAC transactions listed below. In addition, the 80960MC processor can reserve address space on the L-bus for its own use. This section provides the details for each of the following IAC transactions.

- IAC Message (IAC Type 0011_B)
- Register request using a logical address (IAC Type 0010_B)
- Register request using a physical address (IAC Type 0100_B)
- Register request to BXUs on the L-bus (IAC Type 0000_B)
- Identify device order (IAC Type 0111_B)
- Private L-bus space for the 80960MC processor (IAC Type 1111_B)

IAC access type 1111_B reserves address space on the L-bus for the 80960MC processors exclusive use. All addresses of this access type are ignored by the BXU. This allows the 80960MC processor to perform interrupt acknowledge handshakes, etc.

IAC Message (IAC Type 0011_B)

This IAC transaction passes messages to and from the 80960MC processors. Figure 7-6 shows the IAC message address format.

The *IAC identification* field is equal to FF_H to indicate that this is an IAC transaction.

The *module destination* field is separated into two categories: the *unit-identification* bits (six bits labeled UUUUUU), and a *processor-identifier* bit (labeled N). The low order bit in the *module destination* field is not used and can be any value (indicated by the shaded area).

To specify the module destination of a message, a logical identification number is assigned to the six high-order bits of the *module destination* field. The logical identification number, located in the *Logical-ID* register of the BXU, is assigned by software and must uniquely identify a logical module.

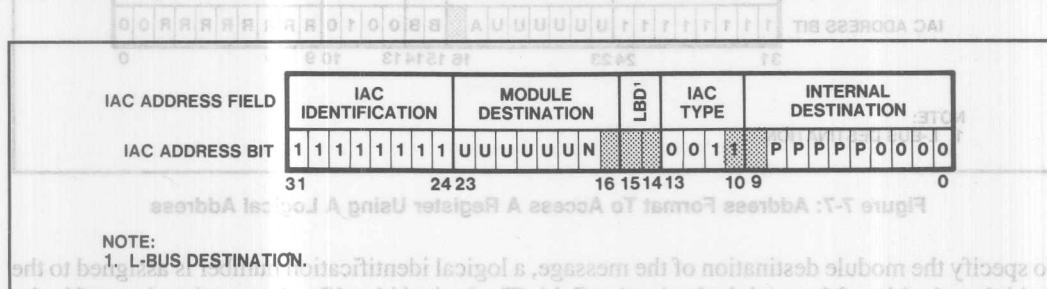


Figure 7-6: Address Format for IAC Message Transaction

Because two processors can reside on the L-bus, the *processor-identifier* bit(N) provides a way to select which of the two processors connected to the destination BXU will receive the IAC message. The serving BXU uses the *Processor-Priority* register associated with destination processor to determine which actions to execute in response to the IAC message.

The *L-bus destination* field is not used.

The *IAC type* field is set to 0011_B to indicate that this is an “IAC message”.

The *internal destination* field is used to communicate the priority of the IAC message. The five bits labeled PPPPP are used by the BXU to determine whether the IAC is accepted or rejected. There are thirty-one levels of priority; PPPPP=0 is the lowest priority and PPPPP=31 is the highest priority. The four low-order bits of this field are set to zero to align the address on a 16-byte boundary because the IAC message can be up to 16 bytes long. The high order bit in this field is not used and can be any value (indicated by the shaded area).

Register Request Using a Logical Address (IAC Type 0010_B)

This IAC transaction allows the 80960MC processor to access a register in a BXU using the logical address of a module. This type of IAC transaction is used primarily in fault-tolerant systems where multiple components operate as a single logical unit (see Part III for complete details). Figure 7-7 shows the address format for this IAC transaction.

The *IAC identification* field is equal to FF_H to indicate that this is an IAC transaction.

The *module destination* field is separated into two categories: the *unit-identification* bits (six bits labeled UUUUUU), and an *access* bit (labeled A). The low order bit in the *module destination* field is not used and can be any value (indicated by the shaded area).

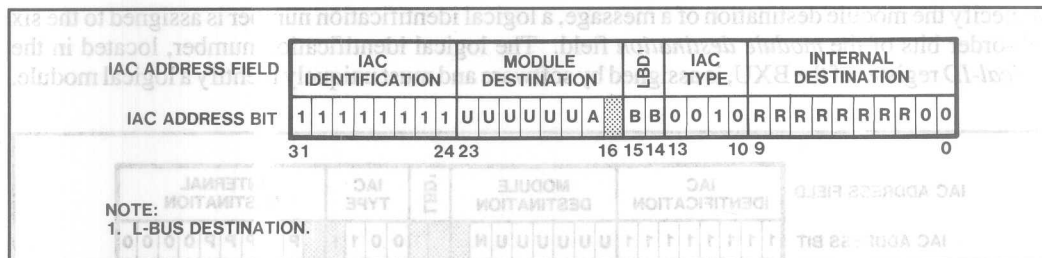


Figure 7-7: Address Format To Access A Register Using A Logical Address

To specify the module destination of the message, a logical identification number is assigned to the six high-order bits of the *module destination* field. The logical identification number, located in the *Logical-ID* register of the BXU, is assigned by software and must uniquely identify a logical module.

The purpose of the *access* bit is to allow testing the fault tolerant circuitry in a single BXU, even though that BXU may be part of a module that consists of two or four identical BXU's with the same logical address. When the access bit is asserted, the *Access-Register* will be used to determine which of the individual agents of the single logical module will respond to the IAC. This bit is used for the testing and start up of fault-tolerant system designs, and is described in detail in Chapter 12.

The *L-bus destination* field (labeled BB) indicates which BXU on the L-bus passes the IAC request to the AP-bus for multiple bus configurations. The BXUs are identified by the ID of the AP-bus to which they are attached. The combination of the *L-bus destination* field and the *module destination* field uniquely identifies one bus agent as the destination of the IAC request.

The *IAC type* field is set to 0010_B to specify that this IAC transaction accesses a register in a BXU using a logical address.

The *internal destination* field specifies a particular command or register of the BXU. The two low-order bits on the AP-bus are equal to zero because all register and command addresses are word aligned. The data written into a register is typically one word long, but can be up to four words long. The data word sent with a command is ignored by the destination BXU. The IAC request must match the register/command size for a valid request. The BXU ignores the byte enable signals on all IAC requests.

Register Request using a Physical Address (IAC Type 0100_B)

This IAC transaction allows the 80960MC processor to access a register in a BXU using a physical address. This type of IAC transaction permits access to bus agents before the final logical configuration is established. Figure 7-8 shows the address format for this IAC transaction.

The *IAC identification* field is equal to FF_H to indicate that this is an IAC transaction.

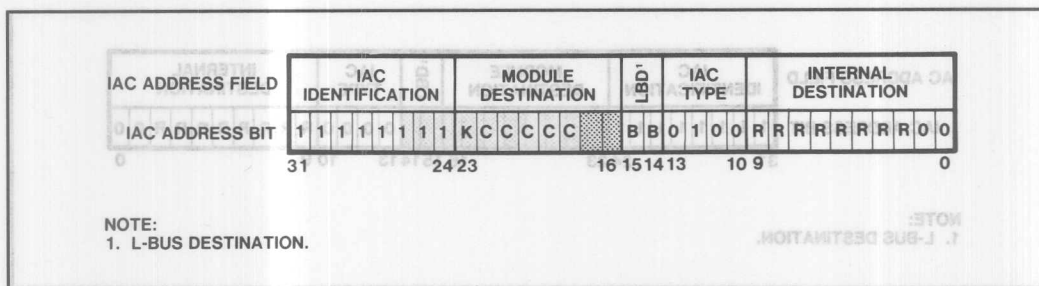


Figure 7-8: Address Format to Access a Register Using a Physical Address

The *Module Destination* field is separated into two categories: a class identification (labeled K), and a component identification (labeled CCCCC). The class ID is set during manufacturing, and is used to distinguish a BXU from other future AP-Bus components. The class ID for a BXU is zero.

The BXU compares the value of the *module destination* field to the value in its *Physical-ID* register (see Appendix A for the description of the *Physical-ID* register). If a match occurs, the BXU responds to the IAC memory-mapped address. The two low order bits in the module destination are not used and can be any value (indicated by the shaded area).

The *L-bus destination* field (labeled BB) indicates which BXU on the L-bus passes the IAC request to the AP-bus for multiple bus configurations. The BXUs are identified by the ID of the AP-bus to which they are attached. The combination of the *L-bus destination* field and the *module destination* field uniquely identifies one bus agent as the destination of the IAC request.

The *IAC type* field is set to 0100_b to specify that this IAC transaction accesses a register in the BXU by using a physical address.

The *internal destination* field specifies a particular command or register of the BXU. The two low-order bits on the AP-bus are equal to zero because all register and command addresses are word aligned. The data written into a register is typically one word long, but can be up to four words long. The data word sent with a command is ignored by the destination BXU. The IAC request must match the register/command size for a valid request. The BXU ignores the byte enable signals on all IAC requests.

Register Request On the L-Bus (IAC Type 0000_b)

This IAC transaction is provided for initialization purposes. It gives the 80960MC processor access to the message-control registers of the BXU on the L-bus. Figure 7-9 shows the address format for this IAC transaction.

The *IAC identification* field is equal to FF_H to indicate that this is an IAC transaction.

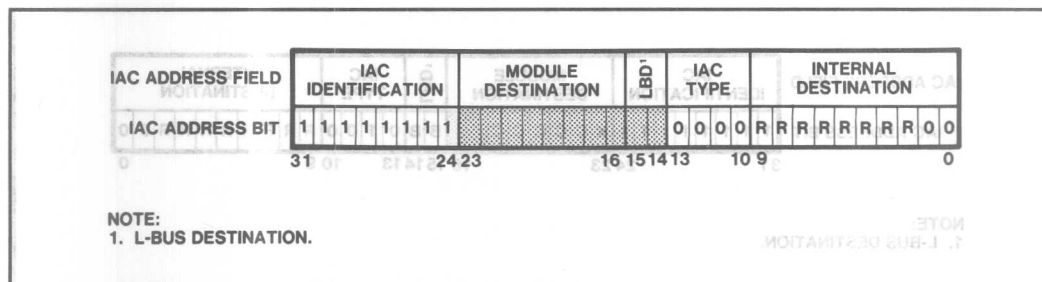


Figure 7-9: Address Format to Access a Register From the L-Bus

The *module* and *L-bus destination* fields are not used because the IAC transaction remains on the L-bus and does not propagate onto the AP-bus. A write operation using this IAC will write data to all BXU's residing on the L-Bus. A read operation is performed only by the message BXU in a multiple AP-bus configuration, unless the access is to the IAC message buffer. In this case, the 80960 will read data from the BXU whose *IAC Message-Data-Valid* bit in its corresponding *Processor-Priority* register is set. Normally, this is the message BXU, but may be another BXU under certain error conditions.

The *IAC type* field is set to 0000_b to specify that this IAC transaction accesses a register of a BXU on the processor's local bus.

The *internal destination* field specifies a particular command or register of the BXU. The two low-order bits on the AP-bus are equal to zero because all register and command addresses are word aligned. The data written into a register is typically one word long, but can be up to four words long. The data word sent with a command is ignored by the destination BXU. The IAC request must match the register/command size for a valid request. The BXU ignores the byte enable signals on all IAC requests.

Identify Device Order (IAC Type 0111_b)

Identify Device Order is a special IAC that supports the initialization of the system. This IAC is used to assign the logical and physical identification values to a particular BXU. Complete details are provided in Chapter 14. Figure 7-10 shows the address format for this type of IAC transaction.

The *IAC identification* field is equal to FF_H to indicate that this is an IAC transaction.

The *module destination* field is not used for this type of transaction. To determine the component destination, this IAC uses the Initialization Identification (*INITID*) pin on the BXU, which is tied to one of the AP-bus address/data lines. This IAC request is transmitted to all the BXUs on a specific AP bus. The individual BXU addressed by this IAC is determined by asserting one of the thirty-two

bits in the first data word. If the asserted \overline{AD} line is physically connected to the BXU's \overline{INITID} pin, that BXU will accept the IAC. All other BXU's will ignore this IAC transaction.

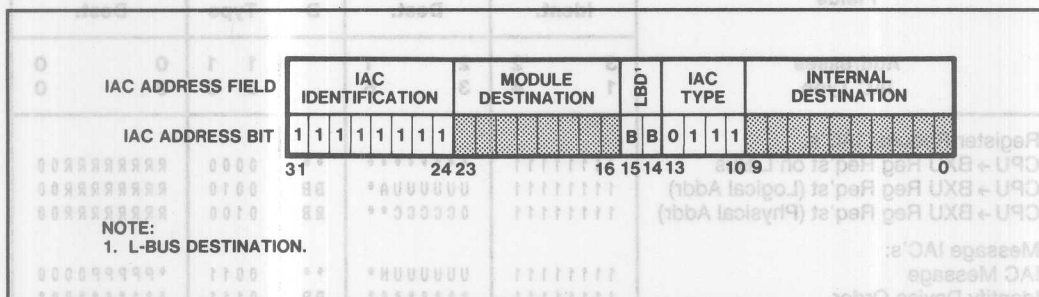


Figure 7-10: Address Format to Identify a Device

The addressed BXU does not send a reply packet to this type of IAC. Consequently, a time-out error is always generated by the requesting BXU. The time-out error causes the requesting BXU to terminate the transaction by generating a BAD-ACCESS reply packet on the AP-bus and asserting the \overline{BADAC} pin on L-bus. The bad-access reply is used to clear the AP-bus pipeline.

The *L-bus destination* field (labeled BB) indicates which BXU on the L-bus passes the IAC request to the AP-bus for multiple bus configurations. The BXUs are identified by the ID of the AP-bus to which they are attached.

The *IAC type* field is set to 0111_b to specify that this IAC transaction is an identify device order IAC.

The *internal destination* field is not used.

To guarantee correct operation, the addressed BXU must either be idle or processing an Identify Device Order IAC from its L-bus. Incorrect operation results if the addressed BXU is processing a memory reference or another type of IAC. This restriction is required because of the special address recognition and register loading that occurs as a result of this IAC.

Summary of IAC Transactions

The 80960 architecture's interagent communications is implemented with two general types of IAC's, register request IAC's and message passing IAC's. The register request IAC's allow a CPU to access any BXU in the system by its logical or physical address. The message passing IAC's facilitate communication between multiple CPU's and bus agents in multiprocessing and fault tolerant system implementations. A summary of the five data fields contained in each IAC is shown in Table 7-10.

Table 7-10: Summary of M82965 IAC Transactions

Fields	IAC Ident.	Module Dest.	LB D	IAC Type	Internal Dest.
Addresses	3 2 2 1	3 6		1 1	0 0
IAC Type	1 4	3		3 0	9 0
Register Request IAC's:					
CPU → BXU Reg Req'st on L-Bus	11111111	*****	**	0000	RRRRRRRR00
CPU → BXU Reg Req'st (Logical Addr)	11111111	UUUUUU*	BB	0010	RRRRRRRR00
CPU → BXU Reg Req'st (Physical Addr)	11111111	0CCCCC*	BB	0100	RRRRRRRR00
Message IAC's:					
IAC Message	11111111	UUUUUU*	**	0011	*PPPPPP0000
Identify Device Order	11111111	*****	BB	0111	*****
CPU Private L-Bus Space	11111111	*****	**	1111	*****

LEGEND:

Neumonic	Definition
A	Access bit
B	L-Bus destination (0-3)
C	Component ident no. (0-31)
N	Processor identifier bit
P	IAC priority (0-31)
R	BXU command or register
U	Unit identification no.
*	Don't care (0 or 1)

AP-BUS PROTOCOL

The Advances Processor Bus (AP-Bus) protocol provides a method to process transactions in an efficient and orderly manner. Figure 7-11 presents an overview on how the AP-bus agents (BXUs) handle transactions packets on the AP-bus. When the bus agent receives signals from the 80960MC processor that require use of the AP-bus, the bus agent arbitrates for control of the AP-bus by using the four arbitration signals (\overline{ARB}_3 , \overline{ARB}_2 , \overline{ARB}_1 , and \overline{ARB}_0). The arbitration occurs during a group of bus cycles called a **time-slice period**.

As a result of the arbitration, the BXUs enter a **grant queue**. The grant queue can hold up to four bus requests from the bus agents. Arbitration is suspended if the grant queue contains four requests. It resumes when there are fewer than four requests in the queue. This queue operates on a First-In-First-Out(FIFO) basis. Thus, the first request to enter the grant queue is the first to exit the grant queue. Once the agent exits the grant queue, it gains access to the AP-bus to perform a bus transaction.

By separating bus transactions into a request packet and a reply packet, transactions can be pipelined to maximize the bus bandwidth. After the bus agent puts a request packet on the bus and enters the transaction into the AP-bus pipeline, it waits for the reply packet. While an agent waits for a reply, other agents can gain access to the bus. The AP-bus protocol permits a maximum of three outstanding requests (transactions pending) in the AP-bus pipeline. Transactions are removed from the pipeline when the corresponding reply packet is received by the requesting agent. A new request packet can enter the pipeline when a slot becomes available.

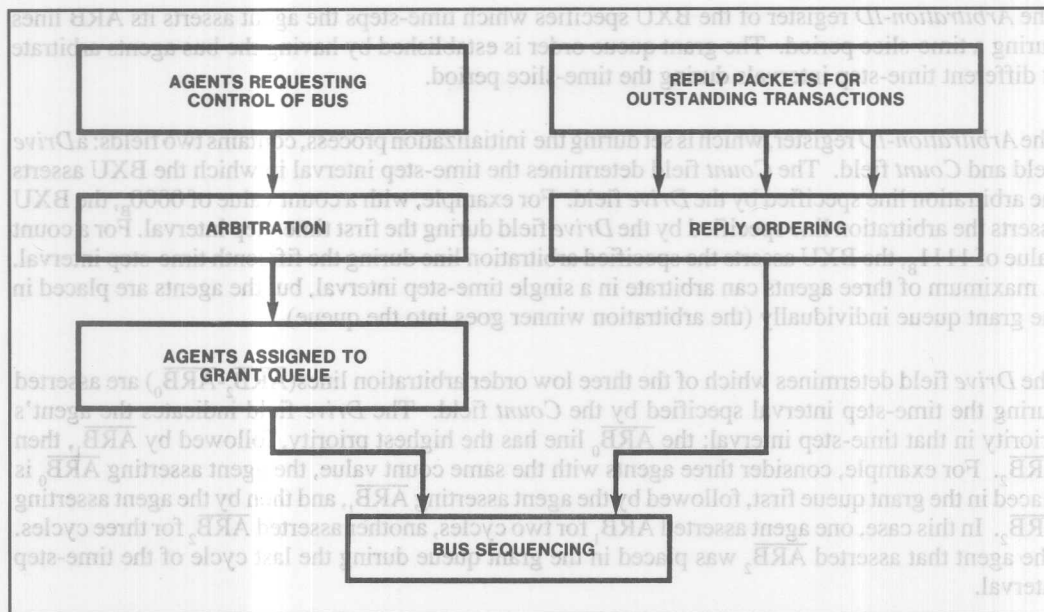


Figure 7-11: AP-Bus Protocol

The reply ordering process controls the order in which the reply packets are sent to the AP-bus. The BXU's bus sequencing control determines when the request packet is placed on the AP-bus and when to send a reply packet. The AP-bus protocol assures that no agent is locked from access to the bus.

Arbitration

The bus agents arbitrate among themselves to obtain access to the AP-bus. The arbitration process uses the \overline{ARB}_3 - \overline{ARB}_0 signals and the *Arbitration-ID* register of the BXUs during a time-slice period to place the bus agents in order in the grant queue (see Appendix A for the description of the *Arbitration-ID* register). The following sections describe the details of the arbitration process and illustrates the process with an example.

Arbitration Process

Arbitration for the AP-Bus is accomplished by using the \overline{ARB}_3 - \overline{ARB}_0 signals and the *Arbitration-ID* register of the BXU during the time-slice period. The \overline{ARB}_0 signal is common to all BXUs on the same AP-bus, the \overline{ARB}_1 signal is common to all BXUs on the same AP-bus, and so forth. All the BXUs on the AP-bus continuously monitor the arbitration signals.

The **time-slice** period consists of a variable number of clock cycles. The time-slice period begins when the BXU's sense that all of the \overline{ARB} signals are deasserted. The time-slice period is divided into a maximum of 16 **time-step intervals** to sequence the requesting agents into the grant queue. When all the requesting agents are placed in the grant queue, the time-slice period ends.

The *Arbitration-ID* register of the B XU specifies which time-steps the agent asserts its \overline{ARB} lines during a time-slice period. The grant queue order is established by having the bus agents arbitrate at different time-step intervals during the time-slice period.

The *Arbitration-ID* register, which is set during the initialization process, contains two fields: a *Drive* field and *Count* field. The *Count* field determines the time-step interval in which the B XU asserts the arbitration line specified by the *Drive* field. For example, with a count value of 0000_B, the B XU asserts the arbitration line specified by the *Drive* field during the first time-step interval. For a count value of 1111_B, the B XU asserts the specified arbitration line during the fifteenth time-step interval. A maximum of three agents can arbitrate in a single time-step interval, but the agents are placed in the grant queue individually (the arbitration winner goes into the queue).

The *Drive* field determines which of the three low order arbitration lines (\overline{ARB}_2 - \overline{ARB}_0) are asserted during the time-step interval specified by the *Count* field. The *Drive* field indicates the agent's priority in that time-step interval: the \overline{ARB}_0 line has the highest priority, followed by \overline{ARB}_1 , then \overline{ARB}_2 . For example, consider three agents with the same count value, the agent asserting \overline{ARB}_0 is placed in the grant queue first, followed by the agent asserting \overline{ARB}_1 , and then by the agent asserting \overline{ARB}_2 . In this case, one agent asserted \overline{ARB}_1 for two cycles, another asserted \overline{ARB}_2 for three cycles. The agent that asserted \overline{ARB}_2 was placed in the grant queue during the last cycle of the time-step interval.

All agents that are not arbitrating during the first time-step interval, but need access to the AP-Bus, assert \overline{ARB}_3 to indicate that additional time-step intervals are required in the current time-slice period. \overline{ARB}_3 will remain asserted as long as any bus agent has not gained access to the grant queue. For example, assume that the *Arbitration-ID* register specifies that the bus agent asserts \overline{ARB}_0 during the fifteenth time-step interval. This agent asserts \overline{ARB}_3 during the first time-step interval and keeps it asserted until the fifteenth time-step interval.

All agents requesting the AP-bus must be ready to arbitrate in the first cycle of a time-slice period to be included in that time-slice period. Any agent that requires the AP-bus after the beginning of the time-slice period must wait until the next time-slice period. The duration of the time-slice period depends upon several factors: the number of agents requesting the AP-bus, the depth of the grant queue, and the count specification in the *Arbitration-ID* register.

During a time-slice period, all of the B XUs are placed in proper order in the grant queue. A maximum of three bus agents can enter the grant queue during any given time-step interval. The grant queue contains four entries. When the grant queue is full, arbitration is automatically suspended and the arbitration lines are held constant. In this case, the time-step interval can be stretched beyond the normal maximum of three cycles by a full grant queue. Arbitration resumes when an agent's request enters a pipeline slot, creating an open slot in the grant queue.

Arbitration Example

The following arbitration example illustrates a typical sequence of events during the arbitration process. The bus agents vie for control of the AP-bus by arbitrating during the time-slice period. As

a result of the arbitration, the agents' bus requests are assigned to the grant queue in the order specified by their respective *Arbitration-ID* registers. The agents enter the AP-Bus pipeline from the grant queue on a first-in first-out (FIFO) basis. When all the agents have been assigned to the grant queue, the time-slice period ends.

The following example shows how seven different bus agents (labeled A, B, C, D, E, F, and G) arbitrate for the AP-bus. The example assumes that the grant queue is empty and that transactions from three agents (X, Y, Z) are in AP-bus pipeline at the beginning of the time-slice period.

Table 7-11 shows the value for the *Arbitration-ID* register of the seven BXU's used in this example.

Table 7-11: Value of the *Arbitration-ID* Register for Agents in the Example

Agents	Arbitration-ID Register						Hexi-Decimal Equivalent for Drive Field*	Hexi-Decimal Equivalent for Count Field
	Drive		Count					
	D	D	C	C	C	C		
A	0	0	0	0	0	0	0	0
B	0	1	0	0	0	0	1	0
C	1	0	0	0	0	0	2	0
D	0	0	0	0	0	1	0	1
E	0	1	0	0	0	1	1	1
F	0	1	0	0	1	1	1	3
G	1	0	0	0	1	1	2	3

NOTE: * Assumes that the two high-order bits are zero.

Figure 7-12 illustrates how the interaction of \overline{ARB}_3 through \overline{ARB}_0 define the time-slice period and time-step intervals.

During clock cycle 1, agents A, B, C, D, E, F, and G are ready to arbitrate. Time-slice period N begins on the next clock cycle, because the four \overline{ARB} lines are deasserted. At this time, the grant queue is empty, and the pipeline, which contains three slots, is full.

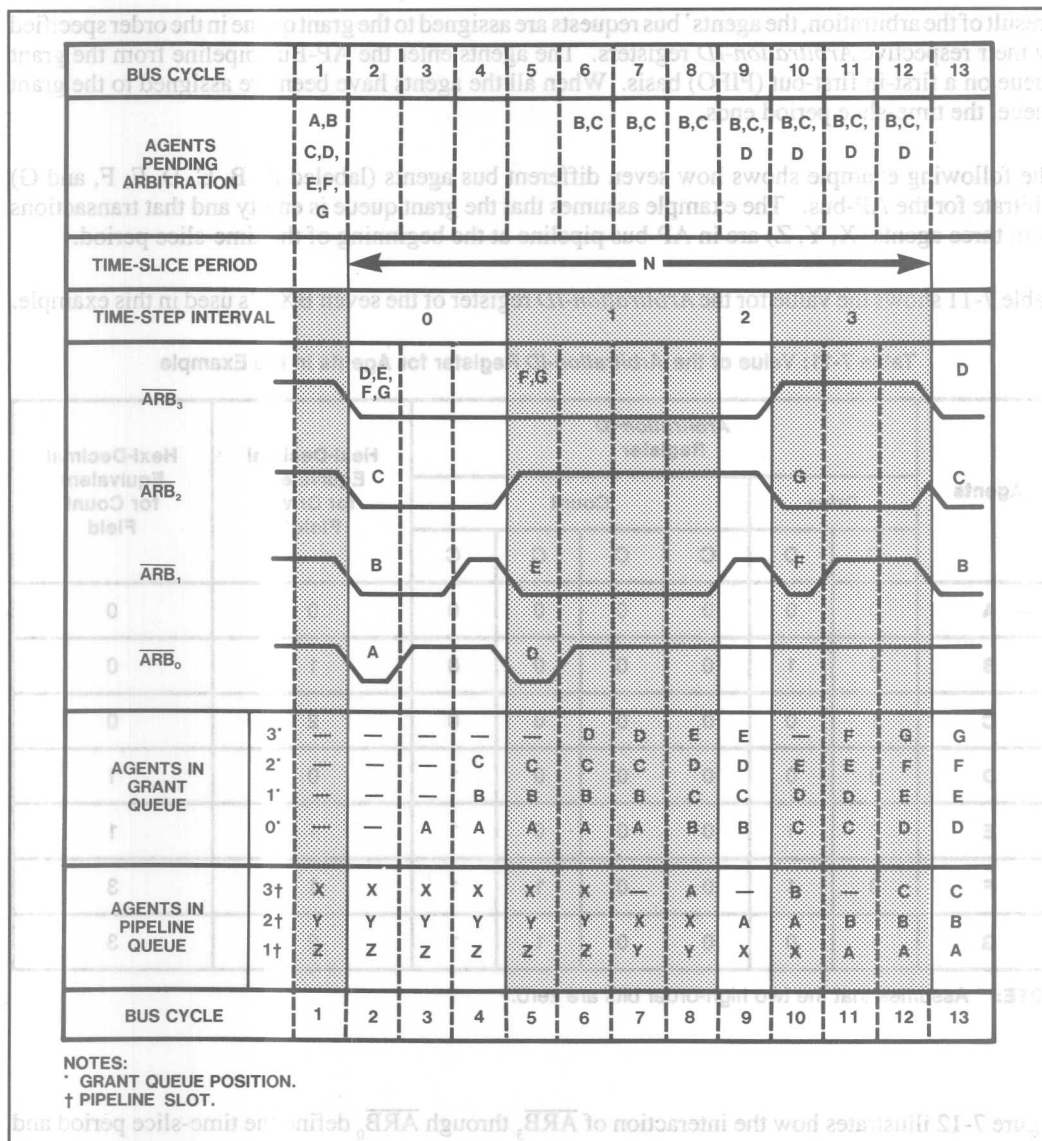


Figure 7-12: Arbitration Example

Time-slice Period N, Time-step Interval 0

The following actions occur during time-slice period N, time-step interval 0.

- Agents A, B, and C arbitrate during this time-step interval because the *Count* field in their respective *Arbitration-ID* registers has a value of zero.
- Agent A is assigned to the grant queue first, followed by agent B, then agent C, because of the settings in the *Drive* field of the *Arbitration-ID* register (agent A asserts \overline{ARB}_0 , agent B asserts \overline{ARB}_1 , and agent C asserts \overline{ARB}_2). Note that these \overline{ARB} lines remain active until the agent is assigned to the grant queue.
- Agents D, E, F, and G assert the \overline{ARB}_3 line because they arbitrate in subsequent time-step intervals. Each agent holds its \overline{ARB}_3 line low until it arbitrates in its specified time-step interval.

Time-step interval 0 ends at the end of clock cycle 4 because all of the agents arbitrating in this time-step interval are in the grant queue.

Time-slice Period N, Time-step Interval 1

The following actions occur during time-slice period N, time-step interval 1.

- Agents D and E arbitrate during this time-step interval because their *Count* field in the *Arbitration-ID* register has a value of one. Thus, at the start of time-step interval 1 agents D and E deassert \overline{ARB}_3 , and assert one of their low order \overline{ARB} lines.
- Agent D is assigned to the grant queue first, followed by agent E, because of the settings in the *Drive* field of their respective *Arbitration-ID* registers (agent D asserts \overline{ARB}_0 , and agent E asserts \overline{ARB}_1). Note that the request from Agent E cannot be immediately placed into the grant queue, because the queue is temporarily full. Consequently, Agent E is assigned to the grant queue after Agent A obtains a slot in the AP-Bus pipeline (a reply packet for agent Z's outstanding request was shown at the end of clock cycle 6 to complete its transaction; thus, opening a slot in the pipeline for Agent A).
- Agents F and G continue to assert their \overline{ARB}_3 lines because they arbitrate in subsequent time-step intervals. Each agent holds its \overline{ARB}_3 line low until it is assigned to the grant queue.
- The example shows Agents B and C requesting an additional access to the AP-bus during clock cycle 6. Because time-slice period N was already in progress when these agents made their second request, they must wait until a new time-slice period begins. Remember, a bus agent must be ready to arbitrate in the first cycle of a time-slice period to be included in that period. These agents are listed in the row labeled *Agents Pending Arbitration* in Figure 7-12.
- The pending transaction for agent Y is completed at the end of clock cycle 8 to open a slot in the AP-Bus pipeline.

Time-step interval 1 ends at the end of clock cycle 8 because all of the agents arbitrating in this time-step interval are in the grant queue.

Time-slice Period N, Time-step Interval 2

The following actions occur during time-slice period N, time-step interval 2.

- No agents arbitrate during this time-step interval because none of the *Arbitration-ID* registers of the requesting agents specify arbitration during time-step interval 2.
- The example shows Agent D requesting an additional access to the AP-bus during clock cycle 9. Because the time-slice period N was already in progress, this agent must wait until a new time-slice period begins. Agent D joins agents B and C in the *Agents Pending Arbitration* row shown in Figure 7-12.
- Agents F and G continue to assert their \overline{ARB}_3 lines because they arbitrate in subsequent time-step intervals. Each agent asserts its \overline{ARB}_3 line until it arbitrates in its specified time-step interval.

Time-step interval 2 ends at the end of clock cycle 9.

Time-slice Period N, Time-step Interval 3

The following actions occur during time-slice period N, time-step interval 3.

- The request from Agent B exits the grant queue and takes the next available slot in the pipeline.
- Agents F and G arbitrate during this time-step interval because the *Count* field in their *Arbitration-ID* registers have a value of three. Thus, at the start of time-step interval 3 agents F and G deassert \overline{ARB}_3 , and assert one of their low order \overline{ARB} lines. Because all of the agents requesting AP-Bus access at the beginning of this time-slice period are either in the AP-Bus pipeline, the grant queue, or in this time-step interval; the common \overline{ARB} line is now deasserted.
- Agent F is assigned to the grant queue first, followed by agent G, because of the settings in their respective *Drive* fields (agent F asserts \overline{ARB}_1 , and agent G asserts \overline{ARB}_2). Note that the request from Agent G cannot be immediately placed into the grant queue because the queue is temporarily full. Consequently, Agent G is assigned to the grant queue after Agent C obtains a pipeline slot, opening a position in the grant queue (the transaction for agent X is completed at the end of the clock cycle 10, opening a slot in the pipeline for Agent C).

Agent G deasserts \overline{ARB}_2 at the end of clock cycle 12. At this point all of the \overline{ARB} lines are deasserted. This marks the end of time-step interval 3 and time-slice period N. Since all of the \overline{ARB} lines are deasserted in clock cycle 13, the agents pending arbitration (Agents B, C, and D) will begin a new time-slice period to arbitrate for access to the AP-bus.

Reply Ordering

The AP-bus protocol supports pipelining of up to three request packets on the AP-bus. An AP-Bus Transaction is made up of a request packet and matching reply packet. The request packets enter a pipeline with three available slots. When the requesting bus agent receives a reply packet in response to its request packet, the transaction is completed. This completion of the request-reply transaction opens an slot in the pipeline for another request packet to enter. **Reply ordering** is the process that determines the order of reply packets on the AP-bus.

The slots in the AP-Bus pipeline are designated slot 1, slot 2, and slot 3. Slot 1 is the beginning of the pipeline and contains the oldest request packet. Slot 2 contains the request packet that has been there the second longest time, and slot 3 contains the most recent request packet.

The reply packets are normally returned in the same order that the request packets were placed in the pipeline. For example, consider the case where the request packet for slot 1 is sent on the AP-bus. When the bus requesting agent receives the reply packet, the transaction is completed, and then removed from slot 1. At this time the request packets in slot 2 and 3 increment one position to allow for a new request packet in slot 3. The next reply packet is in response to the request packet that moved from slot 2 to slot 1.

The ordering of reply packets may be modified by the serving Bus Agent. If the serving BXU needs an extended length of time to send a reply packet (to inhibit a bus timeout), it may defer its place in the pipeline by asserting the Reply Deferral ($\overline{\text{RPYDEF}}$) signal. If $\overline{\text{RPYDEF}}$ is asserted, the request packet currently associated with slot 1 is placed at the end of the pipeline in slot 3. $\overline{\text{RPYDEF}}$ may only be asserted by the serving agent that needs to send a reply packet in response to the request packet in slot 1.

Bus Sequencing

The reply ordering process defines how reply packets are placed in position on the AP-bus. **Bus sequencing** defines how request and reply packets are intermixed on the AP-bus.

To determine the bus activity of the next cycle (N+1 clock cycle), the bus sequencing process samples the $\overline{\text{ARB}}_3$ - $\overline{\text{ARB}}_0$, SPEC_0 - SPEC_3 , and $\overline{\text{RPYDEF}}$ signals in the previous cycle (N-1 clock cycle), and checks the state of the grant queue and pipeline of the current cycle (N cycle), as shown in Figure 7-13.

The following rules govern bus sequencing:

1. If an agent is at the top of the grant queue and the pipeline is not full (less than three transactions), then the agent's request packet is directed to the AP-bus in the next available cycle. The next available cycle refers to the bus cycle following the request packet presently being transmitted on the AP-bus, or the current cycle if there is no request packet on the bus.
2. If the pipeline is full (three transactions in progress), then a reply packet will take the next available bus cycle.

3. If the grant queue is empty and the pipeline is not full, then a reply packet will take the next available bus cycle.

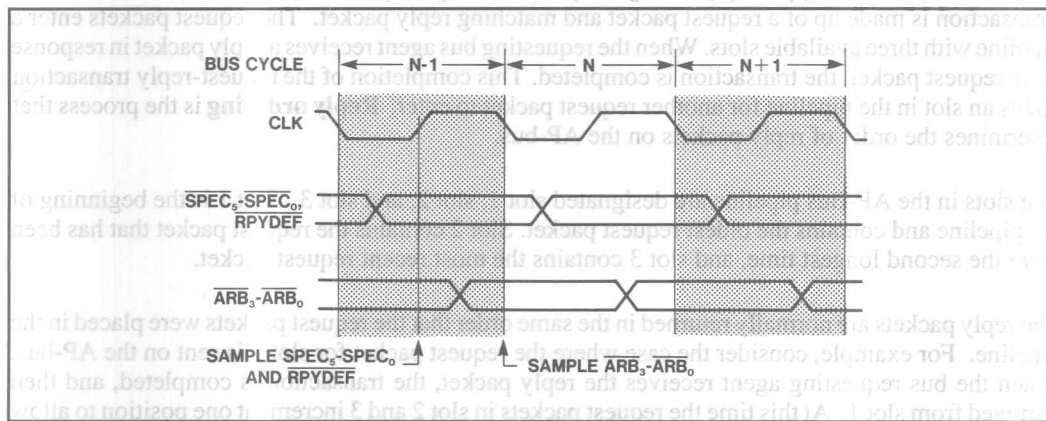


Figure 7-13: Bus Sequencing

The first rule keeps the pipeline as full as possible by giving precedence to request packets over reply packets. In this way, the BXUs maximize the use of the available AP-bus bandwidth.

After the pipeline is full, a reply packet will be placed on the AP-bus, as stated by the second rule. A reply packet completes a transaction to open up a slot in the pipeline. This activity puts the first rule into effect again.

AP-BUS SIGNAL TIMING

The AP-bus signals are referenced to a common system clock (CLK2), which operates at twice the frequency of a bus cycle. A bus cycle defines the period of time for information to be transferred on the AP-bus.

Figure 7-14 shows the relationships between CLK2 and the AP-bus signals. The AP-bus uses three-quarters of the available bus cycle for address and data transmission. The address and data lines are driven on edge D and sampled on edge C. The remaining one-quarter of the bus cycle is allocated for clock skew and signal hold time.

The $\overline{ARB}_3\text{-}\overline{ARB}_0$, $\overline{CHK}_1\text{-}\overline{CHK}_0$, and $\overline{BERL}_1\text{-}\overline{BERL}_0$ signals use the same basic AP-Bus timing concept except that they are offset by one CLK2 bus cycle.

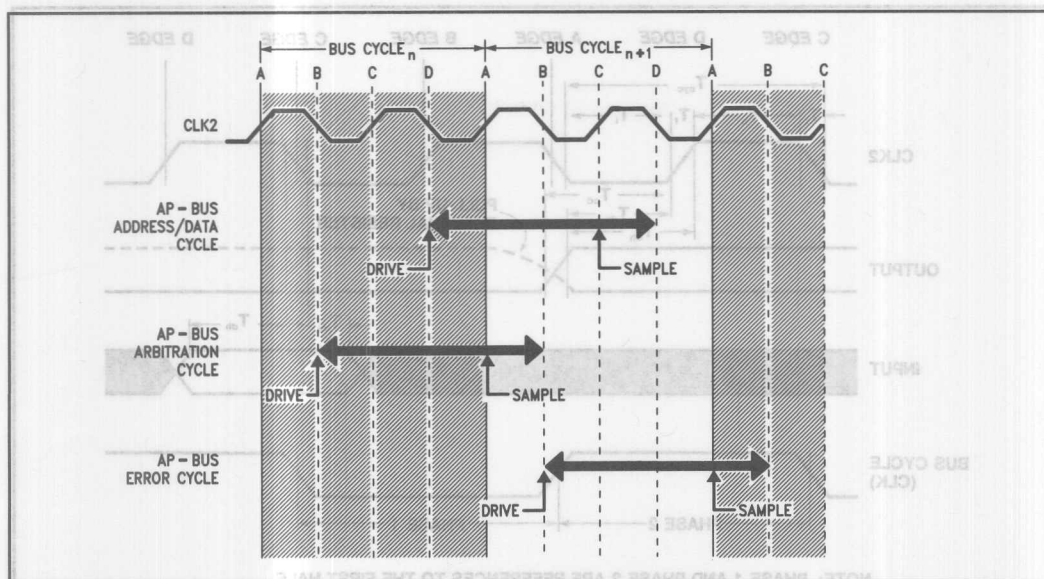


Figure 7-14: AP-Bus Signal Timing

The \overline{ARB} and \overline{CHK} signals lead the \overline{AD} , \overline{SPEC} , and \overline{RPYDEF} signals by one CLK2 cycle, and the \overline{BERL} lines lag by one CLK2 cycle. The \overline{CHK} signals (bus parity signals) are shifted into the next bus cycle to allow time for generation of parity from the internal data that has been transmitted. The \overline{CHK}_1 - \overline{CHK}_0 lines are sampled one clock period after the data is sampled and compared against the internally calculated parity of the received data. The \overline{BERL} lines are similarly shifted into the following bus cycle to allow the BXU's internal circuitry time to verify the received data, in anticipation of an Error Reporting cycle.

General AP-Bus Timing

Most input signals (called group one) on the AP-bus are sampled on the rising edge of CLK2 at the beginning of phase 2 (edge C), as shown in Figure 7-15. The exceptions are the \overline{CHK}_1 - \overline{CHK}_0 , \overline{BERL}_1 - \overline{BERL}_0 , and \overline{ARB}_3 - \overline{ARB}_0 signals (called group two), which are sampled on the rising edge of CLK2 at the beginning of phase 1 (edge A), as shown in Figure 7-16. Regardless of the clock edge, the setup and hold times are the same.

The output signals for group one are driven relative to the falling edge of CLK2 at the middle of phase 2 (edge D), the output signals for group two are driven on the falling edge of CLK2 at the middle of phase 1 (edge B). See Figures 7-15 and 7-16.

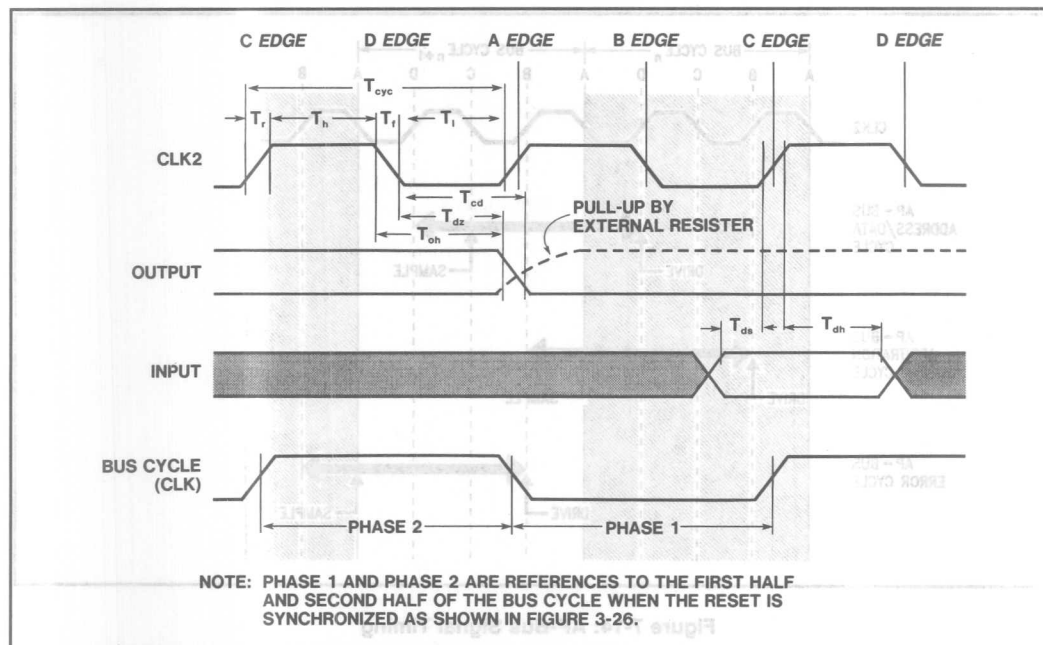


Figure 7-15: AP-Bus Timing for Group One Signals

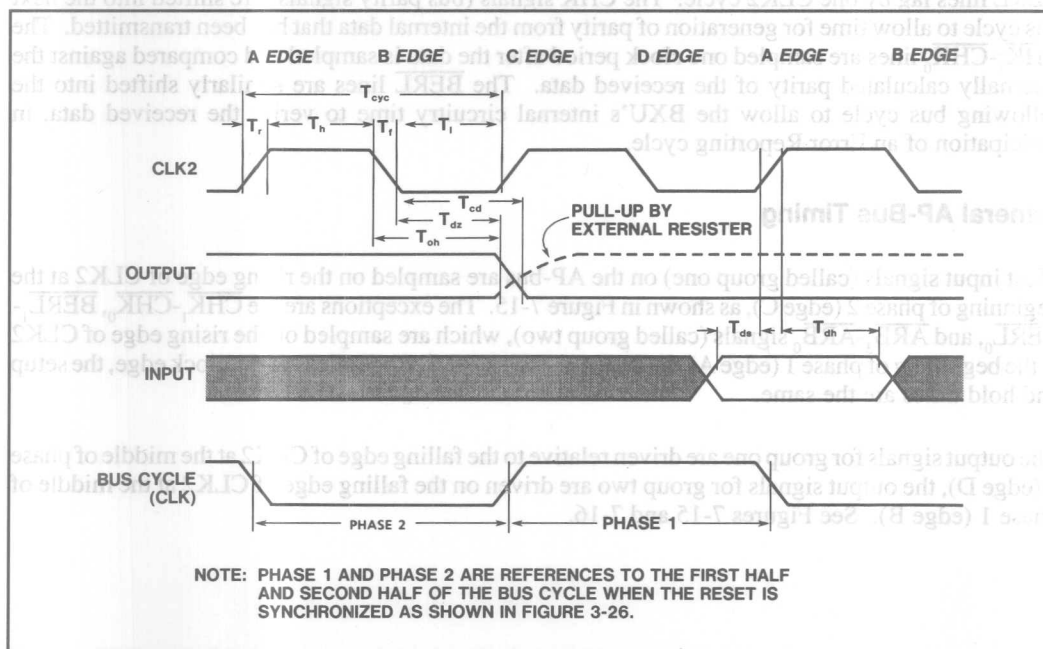


Figure 7-16: AP-Bus Timing for Group Two Signals

AP-Bus Timing Considerations

When designing a system using the AP-bus, the system propagation delays should be considered. This propagation time must allow for settling of ringing, ground shift, and crosstalk; all of which are dependent on board and system materials and design. The following equation can be used to determine the propagation delay, given a specific clock frequency.

$$T_{\text{prop}} = 2T_{\text{cyc}} - [T_f + T_h + T_r + (T_{\text{cd}} \text{ or } T_{\text{dz}}) + T_{\text{ds}} + T_{\text{skew}}]$$

where:

T_{prop} = Propagation delay

$2T_{\text{cyc}}$ = Two system clock periods

T_f = System clock fall time

T_h = System clock high time

T_r = System clock rise time

T_{cd} = Clock to data valid delay time

T_{dz} = Clock to high impedance delay time

T_{ds} = Data setup time

T_{skew} = System clock skew

The system clock skew is defined as follows:

$$T_{\text{skew}} = T_h + T_{\text{oh}} - T_{\text{dh}} \quad \text{where:}$$

T_{oh} = Output hold time

T_{dh} = Input data hold time

SUMMARY

The AP-bus is a synchronous, packetized 32-bit wide bus used to communicate between Agents (BXU's) on behalf of L-Bus residents. Transactions on the AP-bus are executed by using request and reply packets. Special memory-mapped transactions (IAC's) allow the 80960MC processor and BXUs to communicate with each other. The AP-bus protocol defines the arbitration, bus sequencing, and reply ordering. Arbitration is performed by the bus agents as they continuously monitor the arbitration signals, to ensure that all bus agents have access to the bus.

AP-Bus Interface Using the BXU

8

AP-Bus Interface Using the BXU

8

CHAPTER 8

AP-BUS INTERFACE USING THE BXU

The M82965 Bus Extension Unit (BXU) is the key component in building multiprocessor designs with the 80960MC processor family. The BXUs connect to each other in an expandable matrix that can support up to 20 processor and memory modules in a single, high performance system. The BXU increases overall system performance by providing hardware support for local caches, I/O prefetch, message passing, and multiprocessor arbitration.

This chapter describes the operation of the BXU and covers the topics described below.

- An overview of the major logical units of the BXU, which includes a description of the modes of operation and a summary of the registers
- Details of each major logical units of the BXU
- Diagnostic support functions and performance of the BXU
- System configurations

The BXU is also the key to building a fault-tolerant systems with the 80960MC processor family. Through redundant modules, fault-tolerant systems based on the BXU can sustain a single-point failure, and then automatically reconfigure themselves, while application programs continue without disruption. Simply by replicating components, a 80960MC system can be built that can sustain a failure of any single processor, memory array, or bus, and then reconfigure itself. The error detection, isolation, and recovery functions are built in the hardware.

Part III of this manual describes the fault-tolerant design as well as the fault-tolerant aspects of the BXU. This chapter focuses on the multiprocessor design aspects of the BXU.

BXU FUNCTIONAL OVERVIEW

The primary function of the BXU is to connect the Local Bus (L-bus) with the system-wide Advanced Processor bus (AP-bus). In this way, the BXU allows the system to expand incrementally as each new module or system bus is added. Other functions of the BXU include the following:

- Isolation for the AP-bus to allow the construction of large, expandable, module
- Low latency, high-bandwidth memory accesses for sequential data streams used in I/O transfer loops
- Support for multiprocessor systems by reducing a processor's effective demand for system bus bandwidth, allowing the connection of modules to multiple system buses (thereby increasing the total bandwidth available to the module), and providing support registers for communication between processors
- Support for a cache memory on the L-bus.

During normal operation, the BXU is transparent to accesses directed from one bus to the other. For example, if a BXU recognizes an address that lies in its assigned memory address space, it either generates an identical access on the other attached bus or services the request from the cache.

Major Logical Units

The BXU is composed of seven major logical units:

- AP-bus Interface
- L-Bus Interface
- Cache Directory and Control Logic
- Memory Support Logic
- IAC Support Logic
- I/O Prefetch Logic
- Fault Tolerance Support

Individual sections describe the details of operation for each logic unit. The fault tolerance logic unit is described in Part III.

Modes of Operations

The BXU operates in either Processor or Memory mode, and the interpretation of several signals changes depending on the mode of the BXU. Processor mode is intended to support modules that are composed of a 80960MC processor or combinations of 80960MC processors, memory, and I/O devices. Memory mode provides support for memory arrays. The mode of operation is under software control and stored in the LBI Control Register.

Processor Mode

Processor mode provides efficient service to modules composed primarily of 80960MC processors. The BXU supports the cache, prefetch, and IAC message functions. In a pure processor module, there is no need for the BXU to participate in arbitration because it operates as a slave.

In Processor mode, the BXU can also handle modules with a processor and global memory (or master and slave I/O controllers). For this configuration, the BXU can act as both a master and a slave on the L-bus. Although a request flows in either direction through the BXU, the BXU is biased toward handling outbound requests. If there is a significant amount of traffic in both directions, performance can be degraded.

Memory Mode

In the Memory Mode, the BXU acts as a L-bus master in a memory module. No requests are accepted from the L-bus. All requests flow from the AP-bus into the module. When the BXU is in the Memory

Mode, it provides support for memory functions and signal generation. It does not provide the cache interface, prefetch function, or IAC message handling. Memory mode is oriented toward supporting DRAMs and the memory controller.

Table 8-1 summarizes the status and configuration of the major functional blocks and the BXU pins for each mode.

Table 8-1: Summary of BXU Modes of Operation

Item		Mode		Notes
		Processor	Memory	
Logic Unit	AP-Bus Interface	3 Inbound 3 Outbound	3 Inbound 0 Outbound	
	L-Bus Interface	Master or Slave	Master	
	Cache Directory and Control Logic	Available	Not Available	
Support Function	Processor Support	Available	Not Available	1
	Memory Support	Off	On	
	RMW Lock Support	Off	On	2
Dual Function Pins	Mode Dependent Multiplexed Signals	\overline{CR}	DT/ \overline{R}	3
		\overline{CW}	\overline{DEN}	3
		\overline{IAC}_0	ERR	3
		\overline{IAC}_1	FRF	3
		\overline{WY}_0	COR	3
		\overline{WY}_1	MEM/REG	3
		\overline{WD}_0	UNC	3
		\overline{WD}_1	ECC	3

NOTES:

1. The processor support handles the processing of IAC messages received on behalf of the processors in the module.
2. The RMW lock support covers a wide range of possibilities. In Processor mode, the \overline{LOCK} signal is used to indicate outbound RMW-Read and RMW-Write operations. In Memory mode, internal RMW locks are kept to support inbound RMW requests.
3. The pin name for multiplexed signals represents the function of the pin for each particular mode. The pins not listed perform the same function in both modes. See the M82965 data sheet for a complete pin definition.

Register and Command Summary

To provide a flexible interface to various system configurations, the BXU has numerous programmable registers and commands. Complete details of the registers and commands are described in Appendix A.

Initialization and control of the BXU is done by reading and writing to the BXU's registers. Table 8-2 lists all of the registers and commands in the BXU, and table 8-3 provides a memory-map of all the BXU's registers. For completeness, the list includes the registers and commands used in a fault-tolerant design, which are described in Part III.

AP-BUS INTERFACE

The AP-bus interface of the BXU requires no external logic circuits. The AP-bus interface logic performs several functions: address recognition, arbitration, pipeline monitoring, and the AP-bus signal generation. The AP-bus interface logic section of the BXU provides registers that allow software to define its address range. When the BXU recognizes its address from the AP-bus, it responds by either translating the request packet to control signals for the L-bus, or by handling the request internally. For additional request packets, the BXU processes the request packets in the order it receives them. To complete the transaction, the BXU generates the reply packets associated with the received request packets from the AP-bus.

To maximize the use of the pipeline queue, the AP-bus interface provides six buffers, each capable of holding an entire bus transaction. Three buffers are allocated for outbound (to the AP-bus) requests and three for inbound (from the AP-bus) requests.

Memory Address Recognition

The address recognizer logic contained in the AP-bus interface partitions the address space into an L-bus address space and a system bus address space. The BXU does not alter the address that it passes to the L-bus. Except for the upper 16-Mbyte address locations reserved for IAC transactions, the address recognizer is used to match the address on request packets located in 4-Gbyte address range.

The address recognizer consists of two registers: the *AP-Match* (068_H) and *AP-Mask* (06C_H) registers (see Appendix A on for the description of these registers). The *AP-Match* register defines where the L-bus address space begins in the total address space on the system bus, and the *AP-Mask* register defines how much of the available address space on the system bus is mapped to the L-bus. Together, these two registers define a window that maps memory from the AP-bus to the L-bus.

The *AP-Mask* register is used to mask the address bits that select a location in the L-bus address space. The size of the address space mapped to the L-bus determines the number of low-order zeros in the *AP-Mask* register. For example, to recognize 2^N bytes for transferral to the L-bus, the N low-order bits of this register are set to a value of zero. The upper bits from N to 31 are set to a value of one. The size of the mapping window ranges from 256-Kbytes to half of the address space of 2-Gbytes.

Table 8-2: Summary of BXU Registers and Commands

AP-Bus Interface (2)		L-Bus Interface	
Register	Address	Register	Address
PHYSICAL-ID	040 _H	PHYSICAL-ID (LOCAL)	140 _H
LOGICAL-ID	044 _H	LOGICAL-ID (LOCAL)	144 _H
COMPONENT-SPECIFIER (1)	048 _H	LBI-CONTROL	148 _H
ARBITRATION-ID (1)	048 _H	SYSTEM-BUS-ID	14C _H
COM	04C _H	CACHE-CONFIGURATION	150 _H
AP-CONTROL	050 _H	CACHE-TEST	154 _H
FT1	054 _H	LOCAL-BUS-TEST	158 _H
MAXTIME	058 _H	MATCH ₀	160 _H
FRC-SPLITTING-CONTROL	05C _H	MASK ₀	164 _H
TEST-DETECTION	060 _H	MATCH ₁	168 _H
FRC	064 _H	MASK ₁	16C _H
AP-MATCH	068 _H	MATCH ₂	170 _H
AP-MASK	06C _H	MASK ₂	174 _H
		PRIVATE MEMORY MATCH	178 _H
		PRIVATE MEMORY MASK	17C _H
		Command	Address
		INVALIDATE-CACHE	15C _H
		Memory	
		Register	Address
		LOCK	300 _H
		Externally Mapped Registers (Reserved for Memory Controller)	340 _H -37C _H

NOTES:

- (1) The component specifier (read only) and the Arbitration-ID (write-only) are physically two separate registers that share a common address.
- (2) The AP-Bus interface registers are the only registers that can be addressed using access mode.

Table 8-2: Summary of BXU Registers and Commands (cont.)

IAC Message Support				Prefetch		
Register	Address			Register	Address	
PROCESSOR-PRIORITY ₀	000 _H			PREFETCH-CONTROL	240 _H	
PROCESSOR-MESSAGE ₀	010 _H -01C _H					
PROCESSOR-PRIORITY ₁	020 _H					
PROCESSOR-MESSAGE ₁	030 _H -03C _H					

Prefetch Buffer Registers			
Channel	Buffer	Word	Address
0	0	0	3C0 _H
0	0	1	3C4 _H
0	0	2	3C8 _H
0	0	3	3CC _H
0	1	0	3D0 _H
0	1	1	3D4 _H
0	1	2	3D8 _H
0	1	3	3DC _H
1	0	0	3E0 _H
1	0	1	3E4 _H
1	0	2	3E8 _H
1	0	3	3EC _H
1	1	0	3F0 _H
1	1	1	3F4 _H
1	1	2	3F8 _H
1	1	3	3FC _H

Fault Tolerance	
Register	Address
TEST-TYPE	0C0 _H
SPOUSE-ID	0C4 _H
QMR	0E0 _H
MODULE-ERROR-ID	0E4 _H
BUS-ERROR-ID	0E8 _H
ERROR-LOG	0EC _H
ERROR-RECORD	0F0 _H
FT2	0F4 _H
Command	Address
TEST-REPORT	104 _H
PRIMARY-CATASTROPHE	108 _H
SHADOW-CATASTROPHE	10C _H
TERMINATE-PERMANENT ERROR-WINDOW	110 _H
ATTACH-BUS	114 _H
DETACH-BUS	118 _H
SYNC-REFRESH	11C _H

Table 8-3: M80960/M82965 Register Map

Register Function	Internal Dest Addr (H) (10 Bit Field)	Register Contents									
		3	2	2	1	1	0	0	0	0	0
Processor Priority 0	000				PPPP				PWVF		
Processor Priority 1	004										
Processor Message 0	008										
Processor Message 0	00C										
Processor Message 0	010										
Processor Message 0	014										
Processor Message 0	018										
Processor Message 0	01C										
Processor Priority 1	020				PPPP				PWVF		
Processor Priority 1	024										
Processor Priority 1	028										
Processor Priority 1	02C										
Processor Message 1	030										
Processor Message 1	034										
Processor Message 1	038										
Processor Message 1	03C										
Physical-ID	040				KCCCCC						
Logical-ID	044				UUUUUU						
Read = Component-Specifier	048 Read						TTT		TTTSSSSS		
Write = Arbitration ID	048 Write								DDCCCC		
Com	04C	CCCCCCCC			CCCCCCCC	CCCCCCCC			CCCCCCCC		
AP-Control	050								WADRRS		
FT1	054								WACWEB		
Maxtime	058								WTMMMM		
FRC-Splitting Control	05C								WMWPSF		
Test Detection	060								WTFRR		
FRC	064								WRMCTM		
AP-Match	068	BBBBBBBB			BBBBBB				II E		
AP-Mask	06C	MMMMMMMM			MMMMMM				NN		
AP-Mask	070										
AP-Mask	07C										
Test Type	0C0				TTTT	TTTTTTTT			TTTTTTTT		
Spouse ID	0C4					SSSSSSSS					
Spouse ID	0C8										
Spouse ID	0DC										
QMR	0E0								PSWSTWME		
Module Error ID	0E4					CCCCCCCC			SSSSSSSP		
Bus Error ID	0E8					000000BB			SSSSSSSP		
Error Log	0EC		EC		CVTTTTTP	CCCCCCCC			SSSSSSSP		
Error Record	0F0				VTTTTTP	CCCCCCCC			SSSSSSSP		
FT2	0F4								S WFCWBR		
FT2	0F8										
FT2	100										
Test-Report Command	104										
Primary-Catastrophe Command	108										
Shadow-Catastrophe Command	10C										
Term-Perm Err Wind Command	110										
Attach-Bus Command	114										
Detach-Bus Command	118										
Synch-Refresh Command	11C										

[illegible]

8-8-8

The *AP-Match* register defines the L-bus address space within the AP-bus address space. The *N* low-order bits of the *AP-Match* register are ignored, since they are masked by the *AP-Mask* register. This means that the L-bus address space must be aligned on integer multiples of the recognized range.

The *AP-Mask* register should always be set up before the *AP-Match* register to avoid matching on all memory and IAC requests that are placed on the AP-bus. This occurs because 0000_H is the default value of the *AP-Mask* register.

Figure 8-1 shows the basic hardware function provided by the address recognizer. Both the *AP-Match* register and the AP-bus address are masked by the *AP-Mask* register. The results are compared to determine if the address is recognized. In general, the *Recognize* fields of the *AP-Mask* and *AP-Match* registers determine the location of the mapping windows.

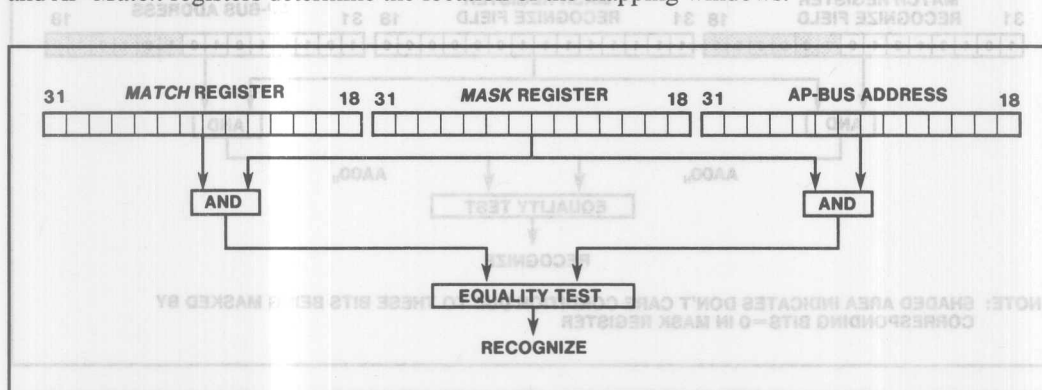


Figure 8-1: Address Recognizer Functional Diagram

Figure 8-2 illustrates how the *AP-Match* and *AP-Mask* registers are used to recognize an address. Assume that the AP-bus address is $AAAA\ AAAA_H$, the *AP-Mask* register is set to value of $FF00\ 0000_H$, and the *AP-Match* register is set to a value of $AA00\ 0001_H$ (the least significant bit in the *AP-Match* register enables the address recognizer assuming that the FT1 and FT2 registers are properly set). The matching logic between the *AP-Match* and *AP-Mask* registers produces a result of 10101010000000_B (or $AA00_H$ assuming bit 17 and bit 16 have a value of zero). Similarly, the matching logic between the *AP-Mask* register and AP-bus address produces a result of 10101010000000_B (or $AA00_H$ assuming bit 17 and bit 16 have a value of zero). Thus, the equality test is true and an address match is accomplished. (See Appendix A for the description of the FT1 register and the FT2 register.)

Guidelines for Recognizer Programming

The programming guidelines listed below should be followed to guarantee consistent, predictable operation of the address recognition mechanism. The third guideline applies to a L-bus address recognizer, which is described in the "L-Bus Interface" section.

1. The *AP-Mask* registers must always contain contiguous values of one in the upper bit positions

- and contiguous values of zero in the lower bit positions.
2. All bits in the *Recognize* fields of the *AP-Match* registers from the position of the highest-order zero in the *AP-Mask* register downward to the lower end of the register should have the value of zero. These bits are masked by the *AP-Mask* register and are ignored by the BXU.
3. The L-bus address recognizers and the AP-bus address recognizers cannot overlap.
4. For multiple BXUs on the same AP-bus, the AP-bus address recognizers cannot overlap.
5. The address recognizers cannot overlap to the IAC address range.

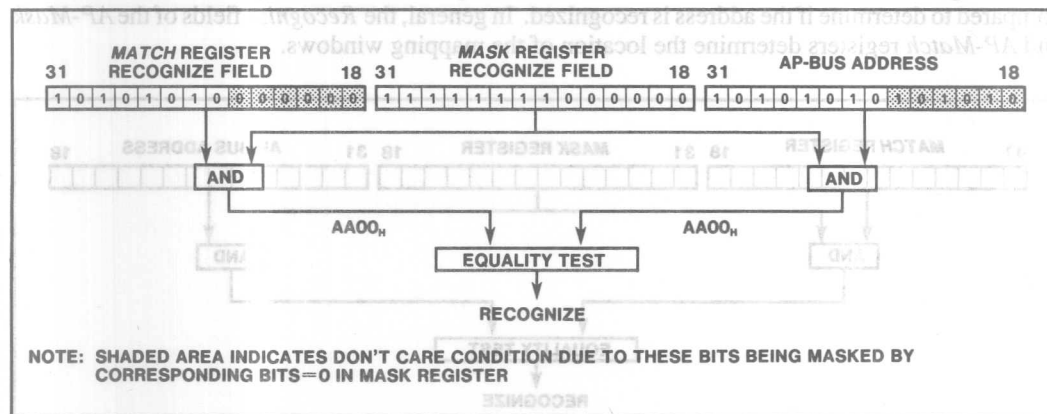


Figure 8-2: Address Recognizer Example

Bus Interleaving

The AP-bus interface supports interleaving of request packets for inbound memory references. Interleaving consists of propagating requests on noncontiguous 16-byte boundaries with 16-byte granularity. The interleaving may be one-way (no interleaving), two-way or four-way. For one-way interleaving, the BXU processes every request that is recognized by its AP-bus memory address recognizer. When the system is two-way interleaved, the system bus address space is effectively divided into two partitions, with addresses 0000_H-000F_H , 0020_H-002F_H , ... assigned to the first partition and addresses 0010_H-001F_H , 0030_H-003F_H , ... assigned to the second partition. A BXU will respond only when an access is recognized by the L-bus address recognizer and is within its assigned partition. Four-way interleaving functions like two-way, but four partitions exist, with the first one consisting of address 0000_H-000F_H , 0040_H-004F_H , etc.

The BXU does not modify the address when it is passed to the L-bus. The interleaving functions are controlled by the *Interleave* Control fields in the *AP-Match* and *AP-Mask* registers. These *Interleave* control fields are separate and independent from the interleaving control for the recognizers in the L-bus interface, which is described in the next section.

When programming the *AP-Match* and *AP-Mask* registers, the address recognizers of the BXUs used for bus interleaving must be set to cover the same address range. The total address space covered

by a mapping window is dependent on the size of the recognized space and the number of ways interleaved. For example, consider a two-way interleaved system with address recognizers set for 256-Kbyte windows. In this case, each BXU actually recognizes only 128-Kbytes, every other 16-byte block in a 256-Kbyte window.

L-BUS INTERFACE

The L-bus interface provides a direct interface between the BXU and the L-bus. The BXU uses different signals on the L-bus depending upon whether it is set in Processor or Memory mode. This section describes the L-bus interface when the BXU is in Processor mode. Operation in Memory mode is described in the "Memory Support" section.

The L-bus interface provides several functions listed below:

- Direct interface between the BXU and the L-bus
- L-bus arbitration logic
- L-bus signal generation
- Forming AP-bus packets from L-bus signals
- Address recognizers that support multiple memory address ranges

When there is more than one BXU on the L-bus, this interface coordinates the activity of the BXUs to provide efficient operation with multiple AP-buses. In this case, the L-bus address recognizers may be set up to support interleaving between multiple AP-buses. This allows the AP-bus activity to be shared among the AP-buses in the system.

In all modes of operation, the BXU always asserts the $\overline{\text{READY}}$ signal on the L-bus during T_a and T_r states. If there are multiple BXUs in a module, they all assert $\overline{\text{READY}}$ during these cycles.

Memory Address Recognition

The L-bus interface has four memory address recognizers and a special one called the Initialization-RAM (INIT-RAM) recognizer. The four memory address recognizers allow portions of the system memory to be interleaved and other portions to be non-interleaved. Moreover, one of the four memory address recognizers maps requests to private memory SRAM on the L-bus.

INIT-RAM Recognition

The INIT-RAM recognizer performs two functions: it allows bootstrapping software to use the cache SRAM as scratch pad storage during system initialization, and it provides a means for executing a memory test on the cache SRAM.

The INIT-RAM recognizer matches an address under the following conditions:

- The value of \overline{AD}_{31} is one.
- The value of \overline{AD}_{30} is zero.
- The *Disable-INIT-RAM* bit in the *LBI-Control* register is set to zero (see Appendix A for the description of the *LBI-Control* register).
- The *Enable-Cache* bit in the *Cache-Configuration* register is set to zero (see Appendix A for the description of the *Cache-Configuration* register).

The Initialize Ram recognizer may remain enabled in parallel with the other recognizers. If the INIT-RAM recognizer remains enabled, the other memory recognizers cannot map addresses to memory locations already covered by the INIT-RAM recognizer. **If a normal memory recognizer and the INIT-RAM recognizer match on the same L-bus request, the operation of the BXU is indeterminate.**

After RESET, the INIT-RAM recognizer is the only memory address recognizer that is enabled in the BXU. Consequently, no memory requests from a 80960MC processor flow to the AP-bus, and thus, the initialization processor boot code and data must be located on the L-bus. The low-order memory addresses are available for PROM storage and the upper addresses are reserved for the INIT-RAM recognizer.

The BXU directs addresses that match the INIT-RAM recognizer towards the local SRAM. For these types of requests, the BXU provides two functions: the sequencing of the \overline{READY} signal and the \overline{WD}_0 - \overline{WD}_0 address bits; and the mapping of the \overline{LAD}_{16} address bit to the \overline{WY}_0 address bit, and the \overline{LAD}_{15} address bit to the \overline{WY}_1 address bit. This provides up to 128-Kbyte of RAM storage for use by the initialization software. In smaller configurations, the INIT-RAM recognizer provides four blocks of storage for every block that is at least 4-Kbytes. See "The Cache Directory and Control Logic" section of this chapter for a description of the address line interface between the BXU and the SRAM.

The timing mode is controlled by the *Timing-Option* bits in the *Cache-Configuration* register. The signal sequencing uses the slow-read, slow-write cache timing, which is the default setting after RESET.

Private Memory Recognition

The private memory recognizer provides the ability to support SRAM on the L-bus as a normal memory, instead of a cache. Private memory is not accessible to other modules, and may only be accessed by agents on the L-bus. The BXU provides registers to establish the address range of the private memory (see Appendix A for the description of the *private memory Match* and *Mask* registers).

The \overline{WY}_0 - \overline{WY}_1 bits are used by the BXU to provide an address for both private memory and cache memory. If cache memory is not used, the private memory can be any size up to 64-Kbytes. The BXU provides the control signals and the address lines for the private memory on the L-bus.

The private memory address recognizer may overlap another L-bus memory recognizer. This is the only case where address recognizers are allowed to overlap. If there is a match in both the private memory recognizer and another memory recognizer, the private memory recognizer has priority and the request is handled from the private memory.

L-Bus and AP-Bus Conversions

During the system operation, the 80960MC processor issues commands that may require use of the AP-bus. For the accesses that require the AP-bus, the BXU converts the signals on the L-bus to the AP-bus request packets. When a reply packet is returned, the BXU converts the packet to the L-bus signals. This section describes special L-bus to AP-bus signal conversions.

Byte and Double-Byte Operations

When the 80960MC processor performs a read operation that asserts a single byte enable signal, the BXU issues a read-byte request on the AP-bus. Similarly, when two byte enable signals

(\overline{BE}_1 - \overline{BE}_0 , or \overline{BE}_3 - \overline{BE}_2) are asserted, the BXU issues a read double-byte request on the AP-bus. For each of these cases, the BXU uses the byte enables signals to generate \overline{AD}_1 - \overline{AD}_0 on the AP-bus. For all other read operations, the BXU issues a read-word(s) request, and the value of \overline{AD}_1 - \overline{AD}_0 are both set to zero.

A read byte request on the AP-bus causes the serving BXU to decode the two low-order address bits to determine which byte enable signals are asserted on the L-bus. Similarly, any read double-byte request on the AP-bus causes the serving BXU to decode \overline{AD}_1 to determine whether if \overline{BE}_1 - \overline{BE}_0 or \overline{BE}_3 - \overline{BE}_2 are asserted. For the read-word(s) request, the BXU asserts the \overline{BE}_3 - \overline{BE}_0 lines.

RMW Operations

A Read-Modify-Write (RMW) operation is initiated on the L-bus by the leading (falling) edge of the \overline{LOCK} signal and terminated by the trailing (rising) edge. The \overline{LOCK} line may stay low for any number of accesses between the start and completion of the RMW operation. This allows accesses by another 80960MC processor on the L-bus, as well as complex indivisible operations by a single processor.

The L-bus interface of the BXU constantly monitors the state of the \overline{LOCK} line. If the previous state of \overline{LOCK} was not asserted (at a high level), and \overline{LOCK} is asserted (at a low level) during the current T_a cycle, and the current transaction is a read operation, then BXU converts the read operation to a RMW-Read request on the AP-bus. After \overline{LOCK} is asserted, subsequent read or write operations flow through the BXU without being converted to RMW operations.

If the previous state of the \overline{LOCK} was asserted, and the \overline{LOCK} line is deasserted during the current T_a cycle, and the current transaction is a write operation, then the current operation is a request for RMW-Write operation. The BXU inserts wait states on the L-bus until all previous requests from the L-bus are completed. Then, the BXU issues the RMW-Write request on the AP-bus. This ensures the integrity of the RMW protocol in a multiprocessor, multiple bus system.

The BXU defines a single, specific use of the $\overline{\text{LOCK}}$ line that is used in communicating with the AP-bus. **Requests from a 80960MC processor that are not to an address recognized by the BXU may use any type of sequence on the $\overline{\text{LOCK}}$ line.** The operation of the BXU is not affected by these $\overline{\text{LOCK}}$ line sequences.

Write-Acknowledge and No-Acknowledgement IAC Replies

The AP-bus protocol requires a Write-Acknowledge or No-Acknowledgement reply for all IAC requests (see section "IAC Support of the BXU" for information on IAC requests). If the IAC is accepted, the serving BXU sends a write-acknowledge reply. If the message buffer is full or the message priority is too low, however, the serving BXU sends a no-acknowledgement reply. The requesting BXU relays the no-acknowledgement information back to the 80960MC processor by asserting $\overline{\text{BADAC}}$ on the cycle after the last data word on the L-bus. Because the BXU requires either an write-acknowledge or no-acknowledgement reply, it inserts wait states on the L-bus until it receives the reply packet on the AP-bus. These wait states are inserted on all IAC requests.

The 80960MC processor cannot distinguish the between a no-acknowledgement reply and a bad-access reply. When the software program receives a no-acknowledgement reply, it typically resends an IAC message until it is accepted. If the receiving BXU was removed from the system because of a fault (described in Part III), a bad-access reply is generated. Because the 80960MC processor cannot distinguish between the two replies, it continues to send the IAC message indefinitely. Consequently, the software program should limit the number of retransmissions to a reasonably small number.

Bad-Access Replies

The AP-bus protocol allows request packets to time-out. For time-out errors, the BXU issues a bad-access reply and asserts the $\overline{\text{BADAC}}$ signal to the 80960MC processor on memory read requests and IAC requests.

The BXU does not issue a bad-access reply for a refused memory write request. When the 80960MC processor reads the location that failed the write operation, the BXU asserts the $\overline{\text{BADAC}}$ signal. The bad-access reply on the AP-bus is transmitted during the first bus cycle of the reply packet. The $\overline{\text{BADAC}}$ signal on the L-bus is asserted on the last T_0 cycle of the transaction. Thus, the L-bus interface of the BXU provides the sequencing to complete the L-bus transaction and signal $\overline{\text{BADAC}}$ if required.

Partial Write Operations

All write requests from the 80960MC processor that require access to the AP-bus flow through the requesting BXU to the AP-bus as full word write-request packets with appropriate byte marks asserted. The serving BXU converts the byte mark signals to $\overline{\text{BE}}_3\text{--}\overline{\text{BE}}_0$ signals at the memory location. The memory controller uses the $\overline{\text{BE}}_3\text{--}\overline{\text{BE}}_0$ signals to write to the appropriate byte(s).

Use of **READY** During Normal Memory Operations

The BXU inserts one wait state for all normal memory operations during the first T_d cycle. The BXU uses this cycle to make the address recognition. This is the only wait state in a normal write operation. For read requests, wait states are inserted until data returns from the AP-bus.

Use of **READY** During Special Memory Operations

The BXU inserts wait states for some special types of memory operations listed below.

- The BXU sends data to the 80960MC processor in response to a RMW-Read request after it receives the entire reply packet. This ensures data integrity.
- For RMW-Write requests, the BXU inserts wait states until all other requests on the L-bus are completed.
- For write requests, the BXU inserts wait states until the write reply packet is received if the *Sequence* bit in the *Match* register is set to one (see Appendix A for the description of the *Match* register).
- The BXU inserts wait states if the slow cache timing options of the *Cache-Configuration* register are selected.
- The BXU inserts wait states for all incoming IAC operations until the reply packet is successfully sent on the AP-bus.

CACHE DIRECTORY AND CONTROL LOGIC

The cache provides high speed local storage for frequently accessed memory locations. By effectively storing the information locally, the cache intercepts memory references and handles them directly without transferring the request to the AP-bus. This action results in lower traffic on the AP-bus and decreased latency on the L-bus, leading to improved performance for a 80960MC processor on the L-bus. It also increases potential system performance by reducing each 80960MC processor's demand for system bus bandwidth, and thereby allowing more processors in a system.

The BXU provides the cache directory and the control logic that implements a cache coherency algorithm. This algorithm ensures that 80960MC processor requests are always correctly serviced, even if there are multiple processors, each with its own cache sharing the same data structures.

The data storage resides on the L-bus in external SRAM components to allow access to a large cache. Some external logic circuitry is required to interface the BXU's control signals to the SRAMs. This section describes the external circuitry and the cache operation.

Overview of Cache Memory Operation

The BXU provides the cache directory, coherency logic, and control signals required to implement a local bus cache. External SRAMs are used for data storage. This approach integrates the most

complex part of the cache design, while keeping the data storage external to the CPU or BXU. Thus, system designs can use the most advanced SRAM technology available.

The cache memory is associated with L-bus requests. In normal operation, the cache memory is controlled by the BXU and is transparent to software. The CACHE signal of the 80960MC processor indicates to the BXU whether a request is "cacheable".

A request is cacheable on the L-bus, if the request meets the following three conditions:

- The request is not an RMW-Read or RMW-Write, or a CPU Write request
- The request is recognized by a L-bus address recognizer
- The CACHE line is asserted (high) during the T_a cycle on the L-bus.

If the above conditions are met, then one of the following actions occurs:

- A CPU read or write request is a cache hit if the requested location matches an address in the cache directory, and data is retrieved from the cache.
- A CPU read request is a cache miss if the requested location does not match an address in the cache directory. Data is retrieved from the system memory, and the cache memory is updated.
- If the CPU write request misses the cache (not an address in the cache) the cache is not updated and the write goes to the memory subsystem via the AP-bus.

The operation of the cache is not dependent on the size of the data transfer and therefore can support partial writes. The BXU does not differentiate between a reference for data or instruction and treats all accesses equally.

Basic Structure of the Cache Memory

Figure 8-3 shows the basic structure of the cache. The cache structure and key cache parameters are defined in the following list.

- A **line**, which consists of 16 data bytes, is the basic unit of data transferred between the cache and system memory. A **line** is referred to a transfer block. A cache hit or miss is determined on a per **line** basis.
- An **address block** is the basic unit for cache addressing. Each **address block** contains the physical address of either eight or four contiguous **lines** of data (128B or 64B).
- A **valid** bit is associated with each **line** within an **address block**. If the **line** present in the cache is valid, then the **valid** bit is turned on.
- A **tag** consists of the address information held in the cache directory. Since many addresses map to a single **address block**, the **tag** information is used to identify the exact memory locations that are currently associated with the **address block**.

to this BXU. This pseudorandom replacement is the only replacement algorithm used. It always specifies the **way** that is used for a new cache entry, even if the other **way** is currently empty.

BXUs are initialized in the same manner as the **way** is updated with a new address. The replacement algorithm is reset by the BXU RESET signal, and every time the *Cache-Enable* bit in the *Cache-Configuration* register is cleared. After reset, the first replacement is to **way0**. The selection of which **way** to replace is toggled on every cacheable access to this BXU.

Write Allocation

The BXU does not perform a write allocation. When a cache miss occurs during a write transaction on the L-bus, the BXU does not attempt to write to the cache. (In a write allocation scheme, an **address block** is allocated for the write operation. Any extra data needed to fill the transfer block in the data array is read from the AP-bus.)

Update Policy

The update policy specifies how main memory is written when the cache is written on the L-bus. The BXU uses a write-through policy. When a cache hit occurs during a write transaction on the L-bus, the BXU writes to the cache memory as well as main memory. Main memory always holds a valid copy of all data locations. The write-through policy implies that the cache cannot off-load write requests from the traffic on the AP-bus.

Coherency

The cache control logic implements a coherency mechanism to guarantee that all 80960MC processor requests result in returning correct and consistent data in systems with multiple caches. The cache control logic constantly monitors all write requests on the AP-bus. Whenever a write is performed on the AP-bus to a location that is also held in the cache (i.e., a cache hit occurs for the particular system bus address), the **line** is marked invalid. Thus, the next time a L-bus request is made to that location, the BXU accesses main memory to retrieve the most recent copy of the data.

Cache Configuration and Control

The configuration of the cache is controlled by two registers: the *LBI-Control* and *Cache-Configuration*. The *LBI-Control* register specifies the interleaving factor and the *Cache-Configuration* register specifies the directory organization and timing options.

There are two primary functions of the *Cache-Configuration* register: the selection of a range of SRAMs for the cache data array, and the selection of the timing of the external data array. These options can incorporate a sophisticated, high performance cache memory, or a simple, lower performance cache memory. Care must be exercised, however, when setting the bits of the *Cache-Configuration* register to avoid setting an invalid combination.

Directory

The BXU supports a two-way set associative cache directory with 64 *sets*. The *Cache-Configuration* register allows the effective directory to be made smaller. The directory storage array, however, always remains the same size. The information stored in the directory array is the same for each *way* within every *set*. It consists of the following items:

- **Tag.** The *tag* field is 20 bits long and consists of the $\overline{AD}_{31} - \overline{AD}_{12}$ signals. The settings in the *Cache-Configuration* register can mask some of these bits for different configurations.
- **Line-Valid bits.** There are eight *line-valid* bits, one bit for each *line* in the *address block*. The settings in the *Cache-Configuration* register can mask some of these bits for different configurations.
- **Tag-Valid bit.** This bit indicates if the *tag* bits are valid. If this bit is set to zero, then the *tag* and *line-valid* bits are ignored.

Operation With Multiple BXUs

A single BXU supports 16-Kbytes of cache memory. When an active module uses multiple BXUs, the BXUs work cooperatively to provide a larger directory and addressing for larger data storage. The best way to view this larger directory is to consider it as having an increased number of *sets*. Thus, a cache managed by two BXUs will have a directory consisting of 128 *sets* instead of 64.

In order to use multiple BXUs, the cache must be set up for interleaving. The number of BXUs must be the same as the interleaving factor. For example, if there is no interleaving, the cache is controlled by a single BXU. If two-way interleaving is used, two BXUs are required, and if four-way interleaving is used, four BXUs share control of the cache.

Coherency

The BXU guarantees that the cache data is consistent with the latest version of memory even in the presence of multiple caches. The coherency mechanism continuously monitors the AP-bus traffic for updates to locations currently in the cache directory.

When the coherency mechanism detects a write request from another BXU or a non-cacheable write request from itself, the address is directed to the cache directory of the BXU. If a cache hit occurs, then the coherency mechanism marks that *line* invalid by resetting the *valid* bit. It is important to note that a *line* (16 bytes) is the unit of validity. Even if a write occurs to a single byte, the entire *line* is marked invalid.

The coherency mechanism is applicable to the cache directory. It guarantees that the BXU cache returns the correct data for various configurations: for systems that use multiple caches, for systems where the processors designate a single data item differently (e.g., some processors designate data as cacheable, and others as not cacheable); or for systems that use two 80960MC processors on a single L-bus.

The coherency mechanism is based upon the write-through update policy. The write-through update policy ensures that the system memory always has the most recent copy of all data structures. The cache memory attached to the BXU never contains the only valid copy of data. This action results in the following conditions:

- System memory **always** has the most recent copy of the system data. In other words, the cache memory is not the only memory location that has the latest version of the data.
- Any access that flows through to the system memory is always guaranteed to receive the latest copy of a data item.
- Any time a processor updates a cache entry, it always causes a write request on the AP-bus, which eliminates any hidden updates.

The coherency mechanism maintains cache data integrity if there are two 80960MC processors on the L-bus. Any L-bus write request that is marked as non-cacheable causes a coherency check in the cache. If the location is present in the cache, the *line* is marked invalid. This allows two processors to swap memory on the same L-bus.

SRAM Interface

The interface between the directory and control logic of the BXU and the external SRAMs consists of six signals. The SRAMs directly interface to the L-bus address lines and the byte enables signals.

\overline{CR} The **Cache Read** signal indicates that the cache SRAMs should drive data onto the L-bus in response to a read request. This open-drain signal can be shared between multiple BXUs controlling a single set of RAMs. This signal can be used to simplify the external logic that generates the \overline{CS} , \overline{OE} , and \overline{WE} signals for the SRAMs.

\overline{CW} The **Cache Write** signal indicates that the cache SRAMs should be written with the data on the L-bus. This open-drain signal can be shared between multiple BXUs controlling a single set of RAMs. This signal can also be used to simplify the external logic that generates the \overline{CS} , \overline{OE} , and \overline{WE} signals for the SRAMs.

\overline{WY}_0 - \overline{WY}_1 **\overline{WAY}_0 (\overline{WY}_0)** - provides the high order address for the SRAMs and is designed to directly drive the SRAM address pins without the need for any TTL buffers. \overline{WY}_0 indicates which one of the *ways* in a directory *set* had a cache hit. **\overline{WAY}_1 (\overline{WY}_1)** indicates whether the access is for the cache or for private memory. If private memory is not used, \overline{WY}_1 can be programmed to either be at a high level or a low level when a cache hit occurs. These open-drain signals remain stable throughout the length of a cache access.

\overline{WD}_1 - \overline{WD}_0

The **Word₁** and **Word₀** (\overline{WD}_1 - \overline{WD}_0) signals provide the word address bits for the SRAMs. These two bits indicate which one of the four words within an address line are accessed. Because the SRAM timing is critical, the address information is generated one cycle early, so that an external latch can be used to hold the information. These open-drain signals change for each word of data transferred.

The BXU asserts **READY** during T_a cycle, T_r cycle, and T_d cycles when data is transferred. This simplifies the latching of the \overline{BE}_3 - \overline{BE}_0 signals for the cache (even though the processor does not sample **READY** during the T_a and T_r cycles).

SRAM Support Logic

The SRAM interface of the BXU minimizes the amount of external logic required to control the SRAM devices although some external logic is required to meet the tight timing constraints of the SRAMs. Since the data sheet specification (T_o) for BXU cache signals exhibit a relatively broad Output Valid Delay external flip-flops and gates are used to achieve better resolution of the \overline{CW} , \overline{WD}_1 , and \overline{WD}_0 signal edges. The address and chip select logic example shown in Figure 8-4 implements systems. This example is referenced in the following cache performance discussions:

Address logic

The address lines can be latched using the \overline{ALE} signal from the 80960MC processor. The \overline{WY}_0 - \overline{WY}_1 signals can provide two more address bits for the SRAM. (This particular example uses only \overline{WY}_0). These bits are driven directly from the BXU because they remain constant for the duration of the cache access. The \overline{WD}_1 - \overline{WD}_0 signals are provided by the BXU. These signals are conditioned by an external flip-flop to provide the fast signal edges required by the SRAMs.

SRAM Chip-Select Logic

The SRAM chip-select signals have two paths, one for a read operation and one for a write operation. For a read operation, all chip-select signals are asserted by the \overline{CR} signal from the BXU. For a write operation, the chip-select signal assertion is dependent upon the byte enable signals and the write timing. The byte enable signals from the 80960MC processor are latched during the address state of the first word. The byte enables for the next word of data are latched every cycle that **READY** is asserted. For this reason the BXU asserts **READY** on all T_a cycles, T_r cycles, and T_d cycles when data is transferred.

In order to ensure the cache is filled properly, the byte enable latch is cleared on read requests. No byte enable information needs to be held during read requests because the data is always returned in full words and the 80960MC processor internally selects the desired data bytes.

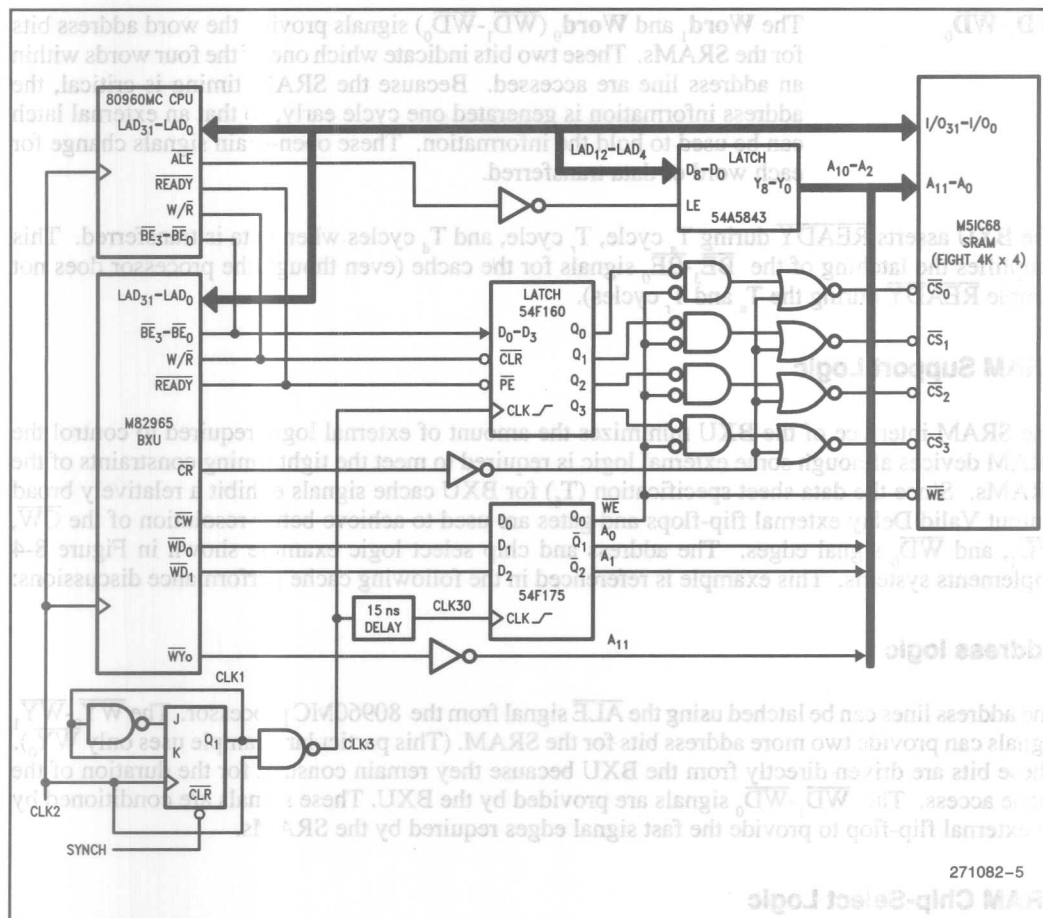


Figure 8-4: Example of External Logic for Cache Interface

SRAM Interface Signal Timings

Figures 8-5 through 8-8 show the timing diagram for the interface signals of the design example shown in Figure 8-4. The diagrams reference the signal timing as 3/6 access timing. This notation means that a one-word read or write operation occurs in three clock (CLK) cycles, and a four-word read or write operation occurs in six clock cycles (one-wait state operation). Similarly, a 4/10 access timing means that a one-word read or write operation occurs in four clock cycles, and a four-word read or write operation occurs in ten clock cycles.

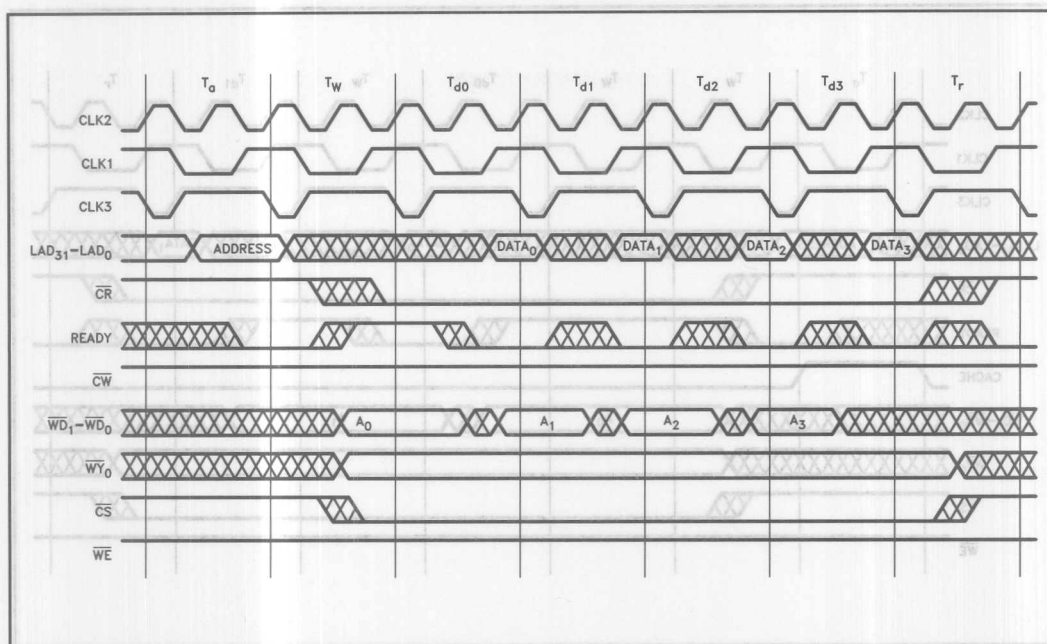


Figure 8-5: Signal Timing for 3/6 Read Access Timing (Fast Read Mode)

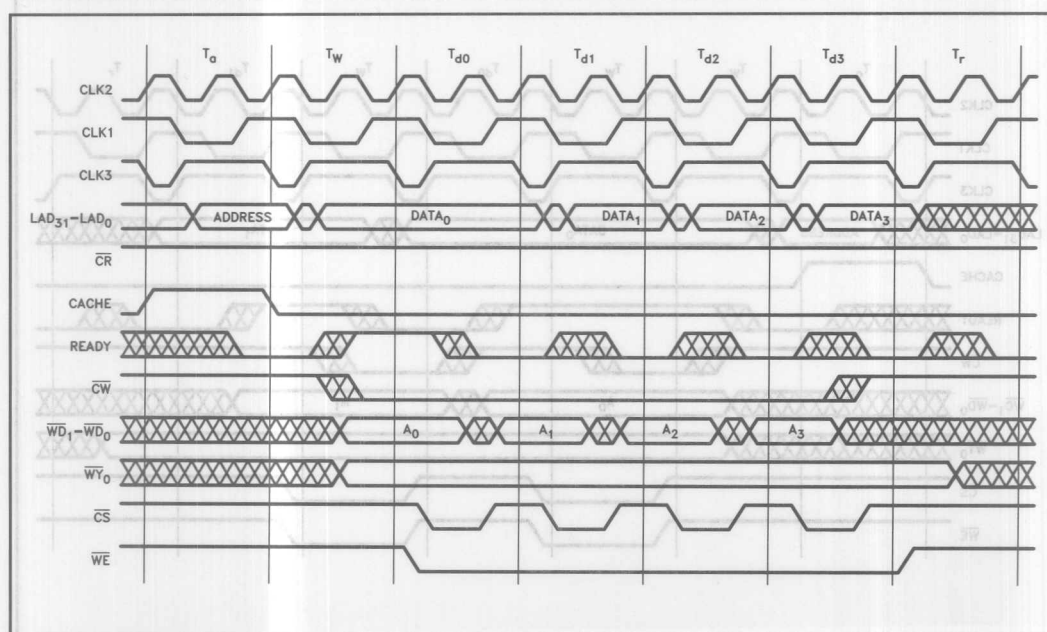


Figure 8-6: Signal Timing for 4/10 Read Access Timing (Slow Read Mode)

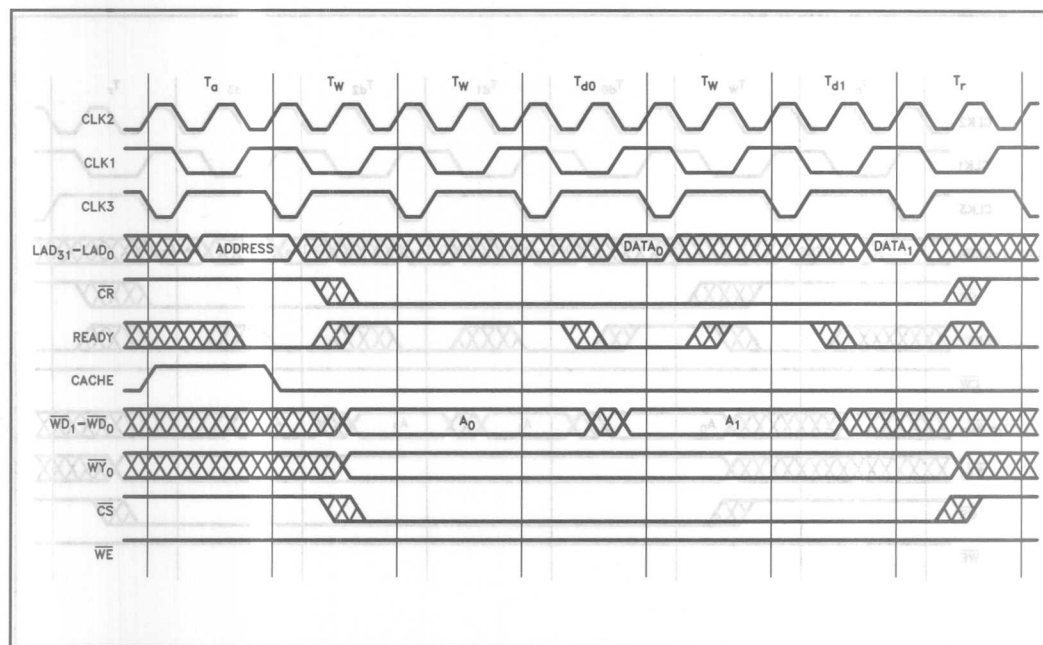


Figure 8-7: Signal Timing for 3/6 Write Access Timing (Fast Write Mode)

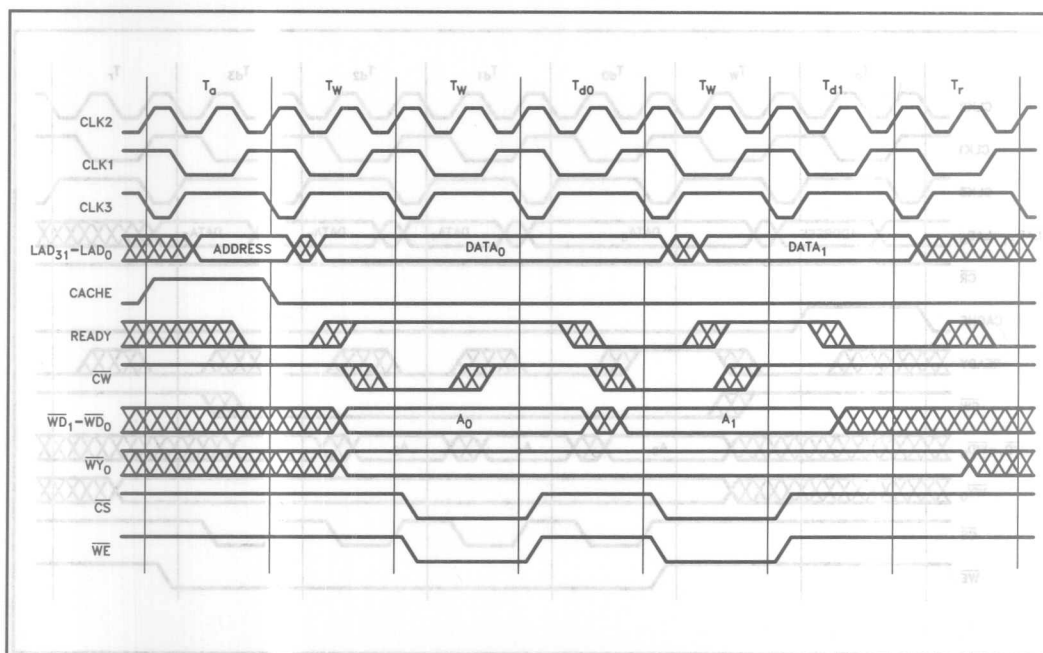


Figure 8-8: Signal Timing for 4/10 Write Access Timing (Slow Write Mode)

Cache Fill Sequence

The cache fill sequence occurs when the 80960MC processor issues a cacheable read request that misses the cache. In this case, the BXU first fetches the missing cache *line* from memory, then returns the requested data to the 80960MC processor.

Upon detecting the miss, the BXU generates the proper 16-byte request on the AP-bus. This request is always aligned on a 16-byte boundary. When the data is returned from the AP-bus, the BXU writes into the cache beginning at word address zero. Figures 8-9 and 8-10 show the signal sequence that performs the fill operation.

Reaction to a Bad-Access Reply

If a request receives a bad-access reply after the cache logic issues a request on the AP-bus to fill a *line* of the cache, then several events listed below occur:

- The *line* of the SRAM cache is filled with invalid data.
- The entire cache directory in the BXU receiving the bad-access reply is invalidated.
- The BXU asserts the $\overline{\text{BADAC}}$ pin at the proper time to relay the information about the bad-access reply to the 80960MC processor.

MEMORY SUPPORT

The BXU provides specific support facilities for memory controllers. The control of a memory module is done jointly between the BXU and a memory controller. The BXU provides the signals for the AP-bus interface, advanced timing information, and access to specific registers maintained

Figure 8-9: Cache Fill 32 Access Timing

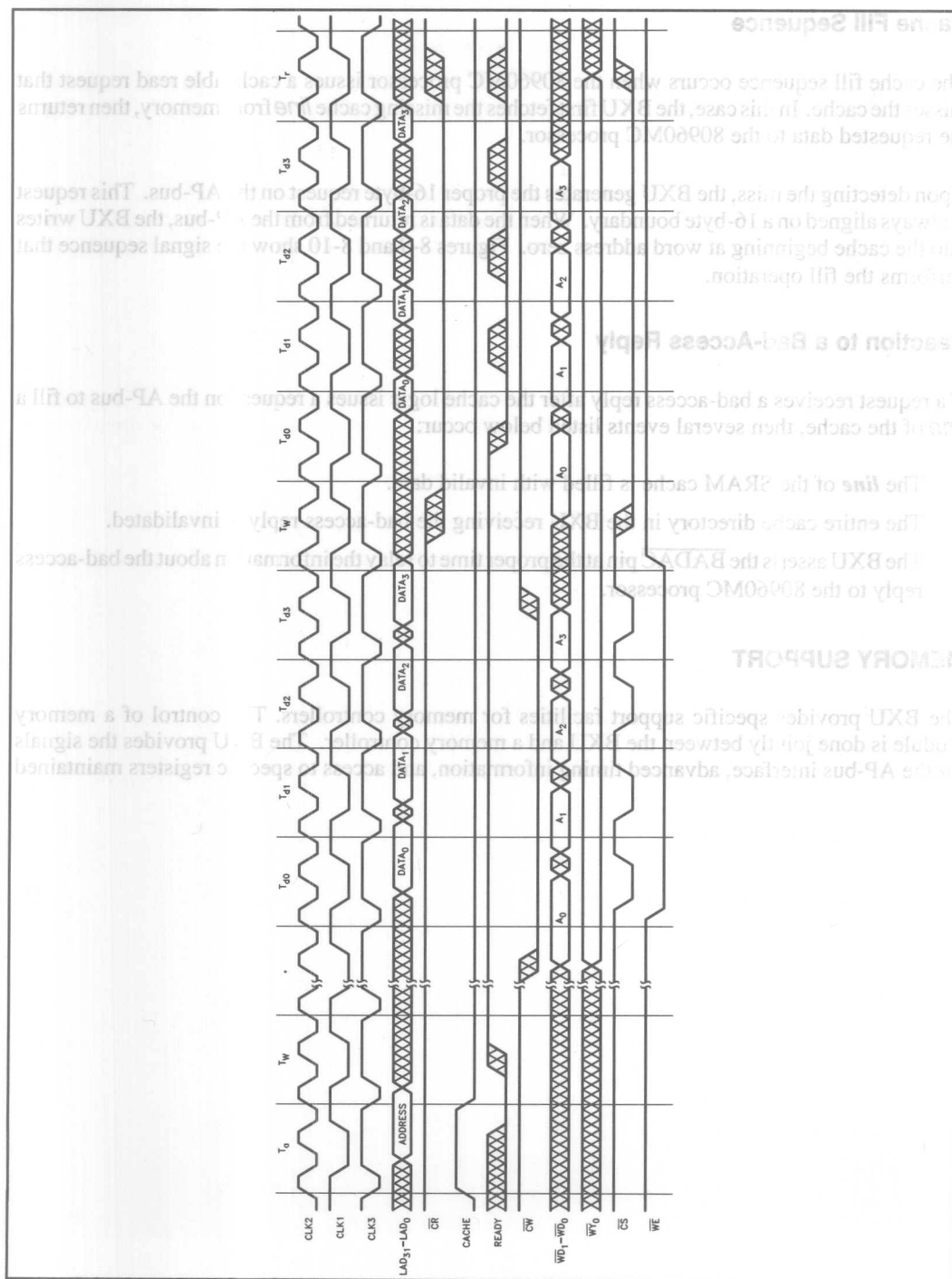


Figure 8-9: Cache Fill 3/6 Access Timing

by the memory controller. The memory controller is responsible for the detailed timing, control, and direct interfacing of the memory components. Normally the memory components will be DRAMS, but other types of memories are supported.

Arbitration

The BXU can act as a primary bus master or a secondary bus master, or it can ignore arbitration completely. The arbitration usage is defined by the *LBI-Control* register, which is configured by software during the initialization sequence. If the BXU is acting as a secondary bus master, one additional cycle is added to the latency for inbound (from the AP-bus) requests.

Normal Operation Support

When operating in the Memory mode, the BXU handles only inbound requests. The cache control logic, I/O prefetch logic, and IAC support logic are disabled and the memory interface signals appear at the multiplexed pins (see Table 8-1). During the T_a cycles on the L-bus, the CACHE signal is used to indicate if the request is cacheable.

Read requests flow through the BXU as quickly as possible. If the BXU is idle, the request is transferred from the AP-bus to the L-bus in 1.5 bus cycles. If the *Fast-Reply* bit in the *Lock* register is set, the data is returned on the AP-bus in 0.5 cycle after it was received on the L-bus (see Appendix A for the description of the *Lock* register).

Write requests are fully buffered before being passed on to the L-bus. After the BXU receives an error free write-request packet, it initiates the write request on the L-bus. When the last data word is accepted on the L-bus, the BXU generates the AP-bus reply packet.

After the BXU starts the L-bus transaction, it completes the write request even if an error report is received. If an error report occurs during a write operation, the write (retry) is repeated on the L-bus.

RMW Locks

In Memory mode, the BXU provides two to four RMW locks with time-outs. Four locks are available if the module is not interleaved with other modules, two locks if it is interleaved. The BXU maps \overline{AD}_6 and \overline{AD}_4 lines to the four *RMW-Lock* bits in the *Lock* register as shown in Table 8-4.

When interleaving occurs, \overline{AD}_4 is used as part of the address recognition for this module. Depending upon the value of this \overline{AD}_4 , the module uses either *RMW-lock* bit₀ and *RMW-lock* bit₂, or *RMW-lock* bit₁ and *RMW-lock* bit₃.

Table 8-4: RMW Lock Map

RMW-LOCK Bit	AD ₆	AD ₄
0	0	0
1	0	1
2	1	0
3	1	1

Other Memory Support Facilities

During initialization, the memory module design may require access to information located in the memory controller. The BXU provides an access by mapping 16 IAC register locations (register address 340_H through address 37C_H) to the memory controller.

When the BXU receives an IAC request to these locations, it maps the request onto the L-bus. When the request is issued on the L-bus, the BXU holds the Memory/Register (MEM/REG) line low during the T_a cycle on the L-bus. This conveys to the memory controller that the request is for its control registers rather than a memory location. While the MEM/REG line is low, all the BXU on that L-bus turn off their L-bus recognizers in order to ensure that the IAC on the L-bus is not accepted by a BXU.

During system operation, the BXU does not translate the address of a request that flows from the AP-bus to the L-bus. The BXU does filter the requests to allow only the requests that are destined for the module. However, the support of various sizes of memory arrays and different interleaving factors require that the memory controller perform some address translation.

The BXU does not indicate when partial (less than a whole word) write operations occur. Thus, the BXU acts like a 80960MC processor on the L-bus to the memory controller.

IAC SUPPORT OF THE BXU

The 80960MC processor uses Interagent Communication (IAC) requests to pass messages to another processor and to read and write the registers of the BXU. This section shows the mechanics of how the IAC facility operates.

IAC Address Recognition

The BXU recognizes an IAC address from the L-bus or AP-bus by comparing the received address to address fields in its registers. The exact match condition is different for each type of IAC.

IAC address recognition is enabled for the L-bus if the *AP-Inactive* bit in the *FT1* register and *Faulty* bit in the *FT2* register are both set to zero. If *AP-Inactive* bit is set to one, but the *Faulty* bit is set to zero, then the BXU accepts IAC requests from the L-bus, but does not propagate them to the AP-bus (the IAC may be handled by a backup bus in fault-tolerant designs). If the *Faulty* field is set to one, then the IAC recognition is disabled.

The match conditions for each type of IAC described in Chapter 7 are listed below.

Local Register Request to BXUs on the L-Bus (Type 0000_B)

Figure 8-11 shows the address fields for the IAC type 0000B. This IAC request is used only on the L-bus and does not propagate onto the AP-bus. Multiple BXUs on the L-bus may recognize a single IAC of this type. They coordinate their response to this IAC by using the **Subsystem Busy** (**SSBUSY**) signal that is common to all the BXUs in the same module. When this signal is asserted, the BXUs accept the requested address, but do not continue with the data cycles. This signal coordinates the reply packets for this type of IAC request. (The **SSBUSY** signal also ensures that the all BXUs are not busy handling IAC requests or performing prefetch operations.)

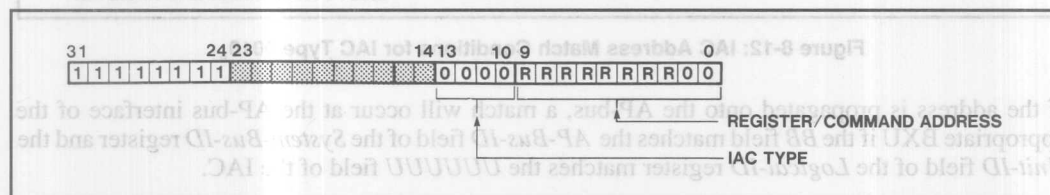


Figure 8-11: IAC Address Match Conditions for IAC Type 0000_B

The Local Register request IAC causes one of the following actions to occur on the L-bus:

- If the processor issues a write request, all BXUs on the L-bus perform the write request.
- If the processor issues a read request to one of the *Processor-Message* registers located at addresses 04_H-07_H, then the BXU with the *Message-Data-Valid* bit set in the *Processor-Priority*₀ register responds to the read request (see Appendix A for the description of the *Processor-Priority*₀ register).
- If the processor issues a read request to one of the *Processor-Message* registers located at addresses 0C_H-0F_H, then the BXU with the *Message-Data-Valid* bit set in the *Processor-Priority*₁ register responds to the read request (see Appendix A for the description of the *Processor-Priority*₁ register).
- Unless one of the above conditions occurs, the message BXU replies to the request. AP-bus₀ is the message bus unless it has failed, in which case AP-bus₁ becomes the message bus. The message BXU is defined by the settings in BXU registers: the *AP-Bus-ID* field of the *System-Bus-ID* register is set to 00_B, and the *AP-Inactive* bit of *FT1* register is set to zero; or the *AP-Bus-ID* field of *System-Bus-ID* register is set to 01_B and *Backup-Bus-Inactive* bit in the *FT2* register is set to one (see Appendix A for the description of the *System-Bus-ID* register).

Register Request Using a Logical Address (Type 0010_b)

Figure 8-12 shows the match conditions for IAC type 0010_b. When the 80960MC processor issues a logical address register request (IAC), a match occurs at the BXU L-bus interface if the IAC's *BB* field matches the *AP-Bus-ID* field of its *System-Bus-ID* register. The BXU processes the request internally if the *Unit-ID* field of the Logical-ID register matches the *UUUUUU* field of the IAC (see Appendix A for the description for *System-Bus-ID* and *Logical-ID* registers). If a BXU on the L-bus's logical ID register does not match the IAC's *BB* field, the IAC request will propagate through the BXU onto the AP-bus.

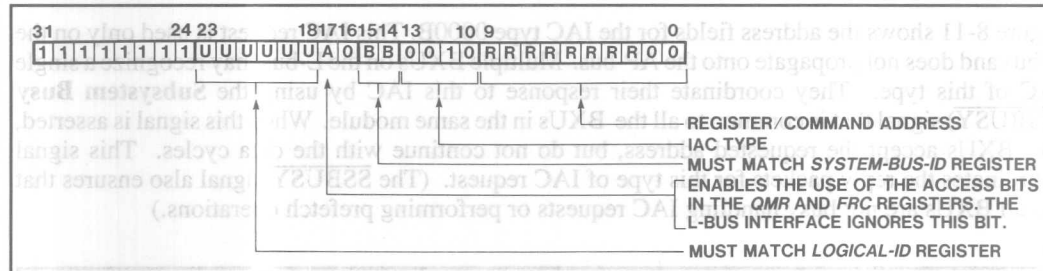


Figure 8-12: IAC Address Match Conditions for IAC Type 0010_b

If the address is propagated onto the AP-bus, a match will occur at the AP-bus interface of the appropriate BXU if the *BB* field matches the *AP-Bus-ID* field of the *System-Bus-ID* register and the *Unit-ID* field of the *Logical-ID* register matches the *UUUUUU* field of the IAC.

The *Access* bit is used to identify a single BXU from others within the same logical module in a fault tolerant system. Chapter 12 describes the usage of this bit.

This IAC address match at the AP-bus interface is only operational if both the *AP-Inactive* bit in the *FT1* register and the *Faulty* bit in the *FT2* register are zero. If either of these bits are a one, this IAC address match is disabled.

Register Request Using a Physical Address (Type 0100_b)

Figure 8-13 shows the match conditions for IAC type 0100_b. When the 80960MC processor issues an IAC physical address register request, a match occurs at the L-bus interface if the IAC's *BB* field matches the *AP-Bus-ID* field of the *System-Bus-ID* register. The BXU processes the request internally on the L-bus if the *Device-ID* field of the *Physical-ID* register matches the *KCCCCC* field of the IAC address (see Appendix A for the description of the *Physical-ID* register). If a BXU on the L-bus's physical ID register does not match the IAC's *BB* field, the IAC request will propagate through the BXU onto the AP bus.

If the address is propagated onto the AP-bus, a match will occur at the AP-bus interface of the appropriate BXU if the IAC's *BB* field matches the *AP-Bus-ID* field of the *System-Bus-ID* register and the *Device-ID* field of its *Physical-ID* register matches the *KCCCCC* field of the IAC.

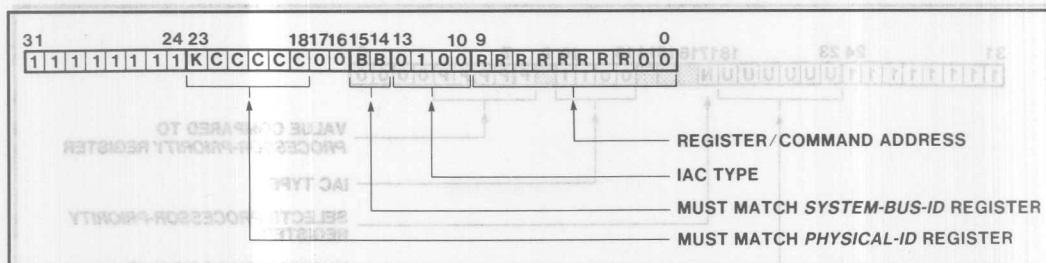


Figure 8-13: IAC Address Match Conditions for IAC Type 0100_b

This IAC address match at the AP-bus interface is only operational if the *AP-Inactive* bit in the *FTI* register is zero. If this bit is one, then this IAC address match is disabled. This IAC is used primarily in fault-tolerant designs.

Identify Device Order (Type 0111_b)

The BXU handles this IAC request if the IAC's *BB* field matches the *AP-Bus-ID* field of the *System-Bus-ID* register, as shown in Figure 8-14. This type of request always propagates on the AP-bus. No reply is generated by the BXU for these requests. See Chapter 14 for more details on the Identify Device Order IAC.

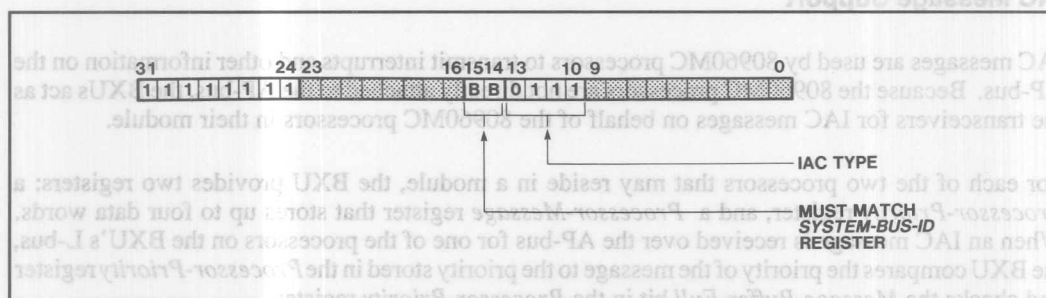


Figure 8-14: IAC Address Match Conditions for IAC Type 0111_b

IAC Message (Type 0011_b)

Figure 8-15 shows the match conditions for IAC type 0011_b. This request type is always handled by the modules message BXU. The BXU processes the request internally on the L-bus if the *Unit-ID* field of the *Logical-ID* register matches the *UUUUUU* field of the IAC. Otherwise, the request propagates onto the AP-bus.

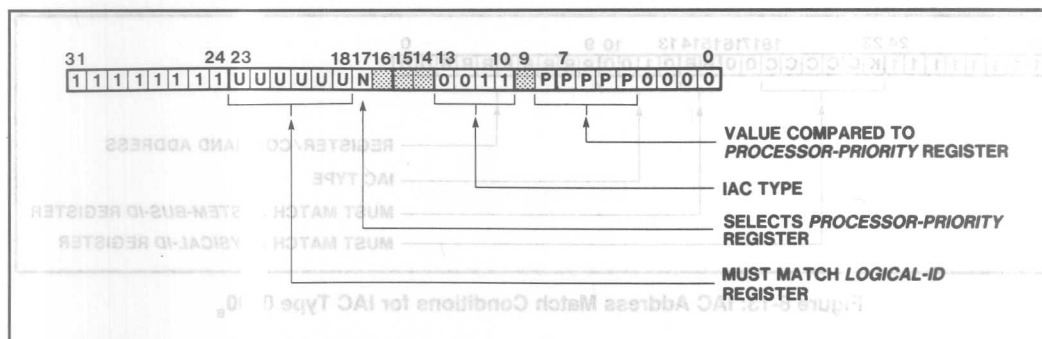


Figure 8-15: IAC Address Match Conditions for IAC Type 0011.

If the address is propagated onto the AP-bus, a match will occur at the AP-bus interface of the appropriate BXU if the *Unit-ID* field of its *Logical-ID* register matches the UUUUUU field of the IAC and the *Message-Buffer-Full* bit in the selected *Processor-Priority* register is zero.

This IAC address match at the AP-bus interface is only operational if both the *AP-Inactive* bit in the *FT1* register and the *Faulty* bit in the *FT2* register are zero. If either of these bits are a one, this IAC address match is disabled.

IAC Message Support

IAC messages are used by 80960MC processors to transmit interrupts and other information on the AP-bus. Because the 80960MC processors are not directly attached to the AP-bus, the BXUs act as the transceivers for IAC messages on behalf of the 80960MC processors in their module.

For each of the two processors that may reside in a module, the BXU provides two registers: a *Processor-Priority* register, and a *Processor-Message* register that stores up to four data words. When an IAC message is received over the AP-bus for one of the processors on the BXU's L-bus, the BXU compares the priority of the message to the priority stored in the *Processor-Priority* register and checks the *Message-Buffer-Full* bit in the *Processor-Priority* register.

Based on the priority comparison and the message buffer availability, the message packet is either accepted or rejected. If the message is accepted, the BXU performs the following actions: it sends a write-acknowledge reply packet on the AP-bus; it notifies the 80960MC processor of the pending message by asserting the *IAC* pin; and it stores the message in the *Processor-Message* register until the 80960MC processor can respond. If the message is rejected, the BXU sends a no-acknowledgment reply IAC. The details of this routine are described below.

Normal Operation

IAC messages are sent using the IAC MESSAGE (type 0011_B). IAC messages are read using the "Local Register Request to BXUs on the L-Bus" IAC (type 0000_B). The sequence during normal operation is described below.

- The BXU on AP-bus₀ acts as the message BXU.
- The 80960MC processors on the L-bus write their current priorities into the appropriate *Processor-Priority* register using IAC type 0000_B. This updates the *Processor-Priority* register in all BXUs in the module at the same time.
- When an IAC message request is received on AP-bus₀, the *Priority* field in the IAC address is compared to the *Priority* field of the *Processor-Priority* register. If the IAC priority is 31 or is greater than the processor's priority, and the message buffer is empty, then the IAC message is accepted.
- If the IAC message is accepted, then three actions occur: an write-acknowledge reply is returned on the AP-bus; the IAC message data is loaded into the *Processor-Message* registers; and the *Message-Data-Valid* bit in the *Processor-Priority* register is set. When the *Message-Data-Valid* bit is set, the BXU asserts the appropriate $\overline{\text{IAC}}$ signal and sets the *Message-Buffer-Full* bit in the *Processor-Priority* register.
- The processor responds to the $\overline{\text{IAC}}$ signal by reading the *Processor-Message* registers using the IAC type 0000_B. The data is provided by the BXU whose associated *Message-Data-Valid* bit is set. The status bits do not change as a result of the read operation. If *Message-Data-Valid* bit is not set by any of the BXUs, no L-bus reply to the CPU is generated and the bus activity is suspended (the bus hangs).
- The 80960MC processor may change the *Priority* field in the *Processor-Priority* register while it is processing the IAC by using IAC type 0000_B.
- Upon completing the IAC processing, the 80960MC processor clears the *Message-Data-Valid* bit in the *Processor-Priority* register. This action automatically causes the *Message-Buffer-Full* bit to be cleared. The 80960MC processor may also update the *Priority* field at this time by using IAC type 0000_B.
- The $\overline{\text{IAC}}$ pin is guaranteed to be deasserted (high) for at least one cycle. Furthermore, the $\overline{\text{IAC}}$ pin is guaranteed to be deasserted by the first cycle after the completion of the L-bus request that cleared the *Message-Data-Valid* bit.
- If the message is not accepted, then a no-acknowledgement reply is returned on the AP-bus. None of the status bits of the *Processor-Priority* register or *Processor-Message* registers are modified.

Interaction With the Registers

The following list shows the details of the interaction of the *Processor-Priority* and *Processor-Message* registers to the IAC message function.

- The *Processor-Priority*₀ register holds the priority of the activity that 80960MC processor₀ is currently executing and the status bits for the processor's message buffer.
- The *Processor-Message*₀ registers (register address 010_H through address 0IC_H) hold four words of data from an IAC message for the 80960MC processor₀. This register block is written

whenever an IAC message is accepted for processor₀. The first data word of the message is placed in register 010_H with the subsequent words going in the next sequential register. The 80960MC processor₀ normally reads the IAC message as a single four-word burst read request.

These registers can be read and written using individual register IAC requests.

The status bits of the *Processor-Priority*₀ register are only modified as a result of an IAC Message or an IAC write request. A normal write operation to the *Processor-Message*₀ register does not change the status bits in the *Processor-Priority*₀ register.

- The *Processor-Priority*₁ register holds the priority of the activity that 80960MC processor₁ is currently executing and the status bits for the processor's message buffer.
- The *Processor-Message*₁ registers (register address 030_H to address 03C_H) hold four words of data from an IAC message for 80960MC processor₁. This register block is written whenever an IAC message is accepted for 80960MC processor₁. The first data word in the message is placed in register 030_H with the subsequent words going in the next sequential registers. The 80960MC processor₁ normally reads the IAC message as a single four-word burst read request.

These registers can be read and written using individual register IAC requests.

The status bits of the *Processor-Priority*₁ register are only modified as a result of an IAC Message or an IAC write request. A normal write operation to the *Processor-Message*₁ register does not change the status bits in the *Processor-Priority*₁ register.

I/O PREFETCH LOGIC

The I/O prefetch unit enhances throughput time for sequential I/O data transfers by providing two 32-byte internal buffers. During typical operation, a memory-mapped address is sent to logic external to the BXU. After the logic decodes the address, it asserts the Prefetch (PFETCH) signal and sends a Start-Channel command (a one-word write request) to the BXU. If the *Prefetch-Control* register is setup to handle prefetch function, the BXU fetches 64 bytes of data based upon the address of the Start-Channel command and fills both 32-byte channels either a word or a byte at a time (see Appendix A for the description of the *Prefetch-Control* register).

When the external logic needs a data word, it reads it from the buffer, thus saving an access to the AP-bus. When the external logic reads the last word in one of the channels, the BXU automatically retrieves another four words of data and places it in this buffer. The prefetch function provides a significant increase in I/O performance because the data requests are handled immediately from the prefetch buffers.

Because the normal operation of the BXU hides the latency of write requests by replying immediately on the L-bus, the prefetch unit only operates on read requests. The BXU does not perform a coherency check on data residing in the I/O prefetch buffers. Consequently, the processor using an I/O prefetch channel must have exclusive access to the data flowing through the prefetch unit.

Signal Definition

The $\overline{\text{PFETCH}}$ signal is used in conjunction with the CACHE and $\text{W}/\overline{\text{R}}$ signals to define the type of request being issued. Table 8-5 defines all of the different request types.

Table 8-5: Prefetch Signal Decoding

$\overline{\text{PFETCH}}$ Signal	CACHE Signal	$\text{WRITE}/\overline{\text{READ}}$ Signal	Operation During T_p Cycle
0	0	0	Read using Prefetch Channel ₀
0	0	1	Start-channel command for Prefetch Channel ₀
0	1	0	Read using Prefetch Channel ₁
0	1	1	Start-channel command for Prefetch Channel ₁
1	0	0	Non-cacheable read
1	0	1	Non-cacheable write
1	1	0	Cacheable read
1	1	1	Cacheable write

NOTE: 0 = Low, 1 = High

When the $\overline{\text{PFETCH}}$ signal is deasserted (high), the current cycle is a normal L-bus cycle. When the $\overline{\text{PFETCH}}$ pin is asserted (low), the current cycle is an I/O prefetch cycle. During prefetch cycles the CACHE pin is used to select one of the two I/O prefetch channels. The $\overline{\text{PFETCH}}$ line must be connected to V_{cc} through an external pull-up resistor when it is deasserted.

Registers and Commands of the I/O Prefetch Unit

The prefetch unit uses two channels that consist of two 16-byte buffers each, which are shown in Table 8-6. The *Prefetch-Control* register is used to specify the channel and other options.

This unit responds to two commands: *Start-Channel₀* and *Start-Channel₁*.

Table 8-6: I/O Prefetch Unit

Channel	Buffer	Address
0	0	3C0 _H -3CC _H
0	1	3D0 _H -3DC _H
1	0	3E0 _H -3EC _H
1	1	3F0 _H -3FC _H

Start-Channel Command

The BXU receives a Start-Channel command before a prefetch channel can be used. A Start-Channel command is defined as a one-word (or less) write request to a prefetch channel. The address in the write request is used as the starting address for the prefetch channel. The data word is not used by the BXU and may be any value.

When the BXU receives the Start-Channel command, the following actions occur:

- The BXUs mark both prefetch buffers in the channel as empty, and compute the starting address.
- If the BXU is involved in the transfer, then the *Active* bit in the *Prefetch-Control* register is set. The BXU issues two prefetch requests on the AP-bus to fill the 32 bytes of data in the channel buffer.
- If the memory locations are in a non-interleaved portion of memory, not every BXU in the module is involved in a transfer. The inactive channels in these BXUs cannot be used for another prefetch transfer. There are only two channels per module, even if a transfer does not use the channel in every BXU.

Every BXU in the module replies to the Start-Channel command when the actions listed above are completed by all the BXUs. (The *SSBUSY* line is used for BXU-to-BXU communication.) When the Start-Channel command is completed, the prefetch requests are initiated, but not necessarily finished.

The combination of the Start-Channel command and the *PFETCH* pin eliminate problems that can arise from reading stale data (data stored in the buffer from a previous cycle). However, the memory being read by the prefetching processor must belong solely to that processor.

Coherency checks are not done on data in the I/O prefetch buffer when the BXU prefetches data beyond the last byte of the last word in the channel buffer as defined in software. This data does not cause a coherency problem because of the following situations:

- The Start-Channel command clears any potential stale data that may have been prefetched during the previous I/O prefetch sequence.

- By using the $\overline{\text{PFETCH}}$ signal, only the software currently using the prefetch channel receives data from the prefetch buffers. Even if a request is to a location already in the I/O prefetch buffer, the BXU accesses memory for the data unless the $\overline{\text{PFETCH}}$ signal was asserted.

Prefetch Data Buffers

Each channel has 32 bytes dedicated for data storage, which are divided into the upper and lower buffers. When the prefetch unit accesses data, it loads either the upper or lower buffer with 16-byte blocks (aligned on 16-byte boundaries). On a read request from the L-bus, the prefetch unit returns the amount of data requested by the 80960MC processor. The prefetch unit never has to cross bank boundaries because the 80960MC processor guarantees that it splits all memory requests that cross 16-byte boundaries into two requests.

Normal Operation

This section describes the setup, startup, and data transfer for the prefetch unit.

Setup

Before an I/O prefetch operation begins, the *Enable* bit in the *Prefetch-Control* register must be set. However, the *Enable* bit does not need to be set every time the prefetch channel is assigned a new address range.

Startup

If the prefetch channels are not assigned statically, the software must allocate a channel to a particular data transfer. The BXU does not provide any support for this allocation process.

The prefetch data must reside totally within a 256-Kbyte partition of memory. The I/O prefetch unit cannot increment across a 256-Kbyte boundary. This restriction eliminates the need for a second address match on the addresses generated by the prefetch unit.

To start the prefetch function, the external logic issues a Start-Channel command while it asserts the $\overline{\text{PFETCH}}$ signal. The prefetch unit uses the address in the Start-Channel command for the first byte to be prefetched.

Data Transfer

A valid request to the prefetch unit must meet the following conditions:

- It must be a standard read request (no write or RMW-Read requests).
- The length of the data request must be a byte, an aligned double-byte (half word), a word, or multiple words.

- It must be recognized by one of the L-bus address recognizers (not the private memory recognizer).
- It must assert the $\overline{\text{PFETCH}}$ signal and have the CACHE signal point at the correct prefetch channel.

Normally the requested data is already present in the channel's data buffer. If this is the case, the prefetch unit returns the data immediately using a 3/6 read access timing (one wait state). If the data is not in the buffer, then the request is held until the AP-bus request fills the buffer (the AP-bus request was issued earlier when the buffer was first emptied).

Data remains in the buffer of the BXU until the last byte in the last word of the buffer is read. The BXU does not maintain the address information used during the prefetch function. When a prefetch request is received, the BXU uses the word address lines ($\text{LAD}_3\text{-LAD}_2$) and size address lines ($\text{LAD}_1\text{-LAD}_0$), and the byte enable signals ($\overline{\text{BE}}_3\text{-}\overline{\text{BE}}_0$). The byte enable signals determine which bytes in the current prefetch buffer are used. The BXU assumes that the request is within the current prefetch buffer. No additional prefetch address checks are done.

When the last byte in a buffer is read, the prefetch unit computes the address for the next 16-byte block and issues an AP-bus read request to this location. This address is based upon the address from the 80960MC processor's prefetch request along with the current interleaving factor specified by the *LBI-Control* register for this address range. The address sent on the AP-bus is within the memory recognition window of the L-bus interface because the incrementer only goes through LAD_{17} and the memory recognizers only look at LAD_{31} through LAD_{18} .

When a prefetch is started, the first data block goes into buffer₀, and the next goes into buffer₁. This pattern is repeated as needed throughout the transfer.

DIAGNOSTIC SUPPORT FUNCTIONS

The 80960MC processor or an external tester can test a BXU by sending bus requests and checking the BXU's reactions to these requests. The BXU's registers and storage for prefetch data can be read and written by IAC requests. This provides sufficient accessibility for test control in most cases. The BXU provides additional diagnostic support for testing in two areas: the cache directory logic, which is described in this section; and the fault tolerance logic, which will be discussed in Chapter 12.

Cache Testing

The BXU provides extensive testing facilities for the cache logic as well as the external SRAM. The cache tests are performed by a 80960MC processor on the L-bus for the external SRAM and the cache directory.

External SRAM

The external SRAM and its associated logic can be tested by using the INIT-RAM recognizer. The INIT-RAM recognizer allows a 80960MC processor to have access to the entire SRAM storage

space. Using this mechanism, the 80960MC processor can directly write and read all the SRAM locations.

BXU Directory Logic

The BXU supports testing the cache directory logic by providing the following facilities:

- BXUs are initialized in the same manner as when the *way* is updated with a new address. The replacement algorithm is reset when the RESET signal is asserted and when the *Cache-Enable* bit in the *Cache-Configuration* register is cleared. After reset, the first replacement is to *way0*. The selection of which *way* to replace alternates on every cacheable access to this BXU.
The replacement algorithm is pseudorandom across a *SET* by toggling the selection of which *way* to replace on every cacheable access to this BXU. This pseudorandom replacement is the only replacement algorithm used. It always specifies the *way* that is used for a new cache entry, even if the other *way* is currently empty.
Diagnostic software can control the cache replacement by asserting the *Cache-Inhibit* bit on all but one L-bus recognizer. In this way only those accesses that are part of the cache test are treated as cacheable by the BXU.
- The BXU provides visibility into the BXU directory lookup table through the *Cache-Test* register (see Appendix A for the description of the *Cache-Test* register).

L-Bus Interface Testing

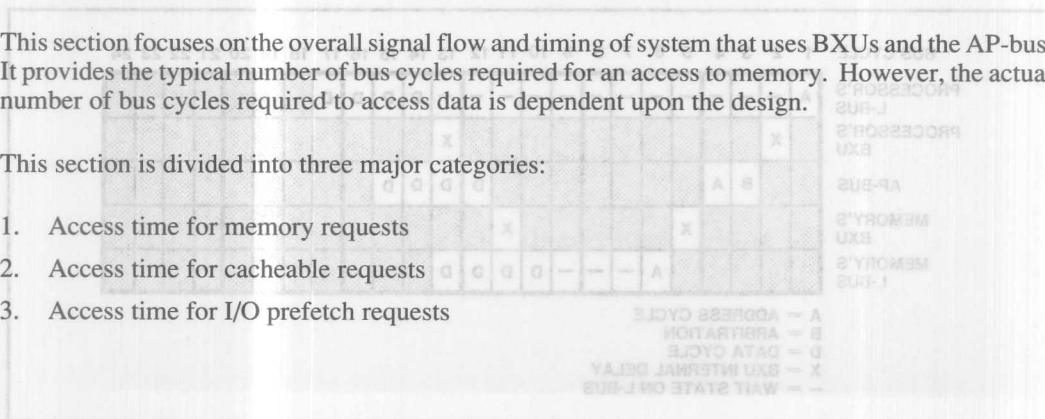
The BXU provides the *Local-Bus-Test* register, which allows the diagnostics to check on the type of recognition that was performed on the previous L-bus access (see Appendix A for the description of the *Local-Bus-Test* register).

BXU TIMING

This section focuses on the overall signal flow and timing of system that uses BXUs and the AP-bus. It provides the typical number of bus cycles required for an access to memory. However, the actual number of bus cycles required to access data is dependent upon the design.

This section is divided into three major categories:

1. Access time for memory requests
2. Access time for cacheable requests
3. Access time for I/O prefetch requests



Memory Requests

Table 8-7 shows the access time for memory requests. Whenever the request flows onto the AP-bus, the access time is dependent on bus loading and access contention. Consequently, the number of bus cycles show the typical access for a BXU and an access for a loaded system. The notation indicates the number of bus cycles required for a one-word and four-word access.

Table 8-7: Memory Request Access Time

	BXU	Loaded
READ	14/17	21/24
WRITE	3/6	3/6
RMW-READ	14/20	21/27
RMW-WRITE	3/6*	3/6*

NOTE:

* Assumes all previous L-bus requests are completed.

Notation: 1 word/4 word access.

Outbound Read Requests

Figure 8-16 shows how the signals propagate from the processor, to memory, and back to the processor on a clock-by-clock basis for a typical read operation. When a 80960MC processor or the cache logic on the L-bus issues an outbound read request, the BXU accepts the base address and the size of the request from the L-bus, but does not acknowledge the request until the data is returned to the 80960MC processor. Hence, during read operations that flow on the AP-bus, the BXU inserts wait states on the L-bus until the read operation is completed.

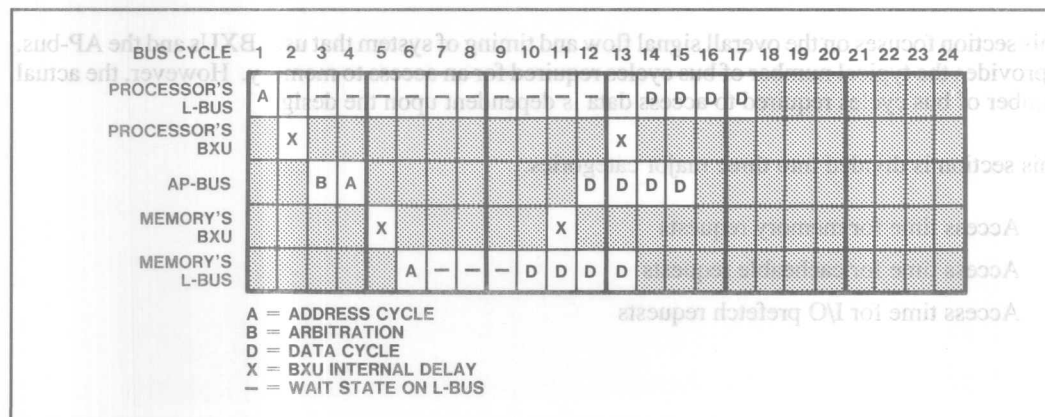


Figure 8-16: Typical Read Timing Diagram

An outbound read request is always the last request in the internal request queue of the BXU. When the read data arrives on the AP-bus, the data flows through the BXU onto the L-bus to improve performance.

Outbound Write Requests

Figure 8-17 shows how the signals propagate from the processor, to memory, and back to the processor on a clock-by-clock basis for a typical write operation. When the 80960MC processor on the L-bus issues an outbound write request, the BXU accepts the base address, the size of the write request, and the data from the L-bus.

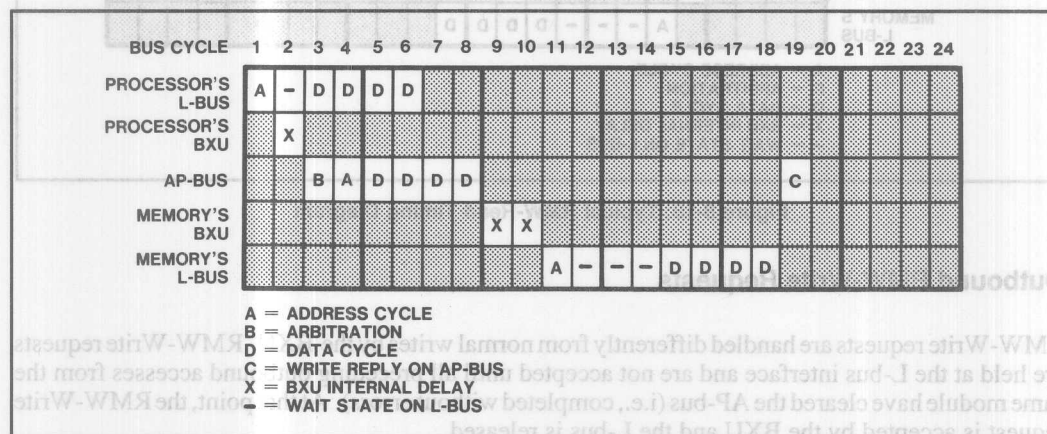


Figure 8-17: Typical Write or Cache Write (Hit or Miss) Timing Diagram

Write requests are immediately acknowledged on the L-bus, thereby releasing the bus and allowing the processors to proceed. A BXU can buffer up to three outbound write requests. If the write request encounters a bad-access error on the AP-bus, the BXU ignores the bad-access reply because the write was already acknowledged on the L-bus. When the 80960MC processor reads the location that failed the write operation, the BXU asserts the BADAC signal. The BXU does, however, respond to all other errors on write operations (e.g., parity error).

Write requests are fully buffered at the destination of the request. This means that all write operations are indivisible with respect to read operations, and that write operations are either completed or not completed at all (aborted).

Outbound RMW-Read Requests

Figure 8-18 shows how the signals propagate from the processor, to memory, and back to the processor on a clock-by-clock basis for a typical RMW-Read operation. RMW-Read requests respond like normal outbound read operations except for buffering the data. When data returns on the AP-bus, the BXU buffers the entire reply packet to ensure that it is correct before transferring

it on the L-bus. Thus, RMW-Read requests are indivisible and never return inconsistent data. The RMW-Read timing is extended if there is contention for the RMW lock.

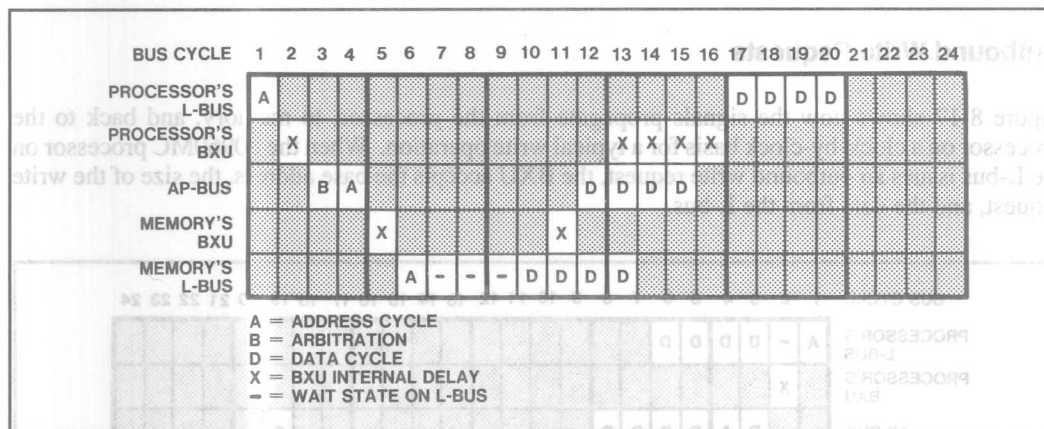


Figure 8-18: Typical RMW-Read Timing Diagram

Outbound RMW-Write Requests

RMW-Write requests are handled differently from normal writes by the BXU. RMW-Write requests are held at the L-bus interface and are not accepted until all preceding outbound accesses from the same module have cleared the AP-bus (i.e., completed without errors). At that point, the RMW-Write request is accepted by the BXU and the L-bus is released.

Access Time for Cacheable Requests

The access times shown in Table 8-8 assume that the cache *Timing-Option* field in the *Cache-Configuration* register is set for a fast-read, fast-write timing option.

Table 8-8: Access Time for Cacheable Requests

	BXU	Loaded
Read Hit	3/6	3/6
Write Hit	3/6	3/6
Read Miss	19/22	26/29
Write Miss	3/6	3/6

Notation: 1 word/4 word access.

The BXU provides one wait state for each cache access. There is no penalty for a miss on a write operation because the request is buffered inside the BXU and the AP-bus access time is hidden.

Figure 8-19 shows the typical timing for a cache read miss. In this example, the cache must be filled before the data is returned. Thus, the BXU always performs a four-word access to memory on read misses. The bus-to-bus connection adds two cycles (one each way), and the BXU adds two cycles (one each way) for moving the data through the chip.

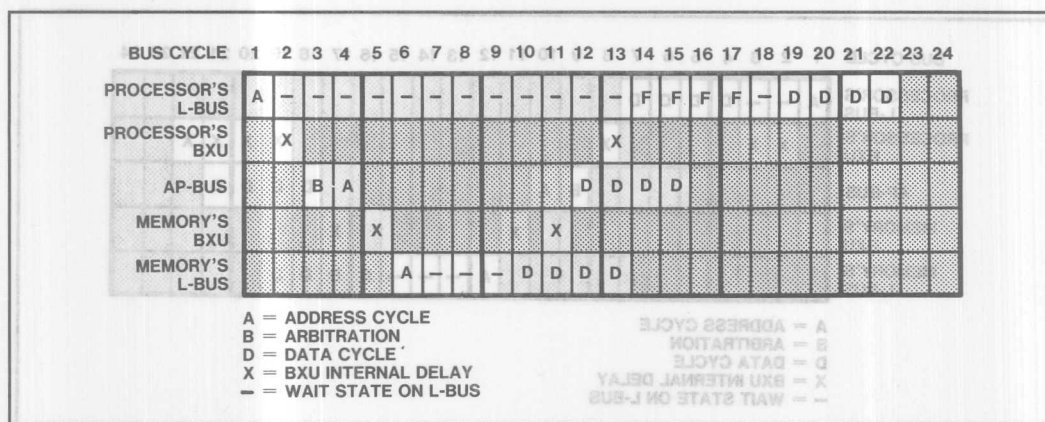


Figure 8-19: Typical Cache Read Miss Timing Diagram

IAC Requests

Table 8-9 shows the access time for IAC requests. All IAC requests are held until the AP-bus reply packets return to the BXU. This action allows the BXU to relay information to the 80960MC processor about the type of reply packet (write-acknowledge, no-acknowledgement, or bad-access). The access time may be different depending on whether the request is sent to a L-bus BXU, or whether the BXU is busy processing other requests.

Table 8-9: IAC Request Access Time

	BXU	Loaded
IAC Read	14/17	21/24
IAC Write	14/17	21/24

Notation: 1 word/4 word access.

I/O Prefetch Request

The BXU contains prefetch logic for two I/O channels. Each channel consists of two 16-byte buffers. The prefetch buffers have to be setup before an I/O transfer. A prefetch read operation is issued on

the AP-bus whenever the last word of the previous prefetch buffer is read. Hence, a prefetch read operation is the only type of read operation that can be intermingled with, and followed by a write request in the AP-bus pipeline queue (see description of outbound read requests).

Figure 8-20 shows the timing diagram for an I/O prefetch read operation. The BXU accesses data in the prefetch buffers in 4/7 access time, assuming that a typical AP-bus access is 17/20. An I/O prefetch channel in a single AP-bus system can transfer data at a rate of 25-Mbytes per second.

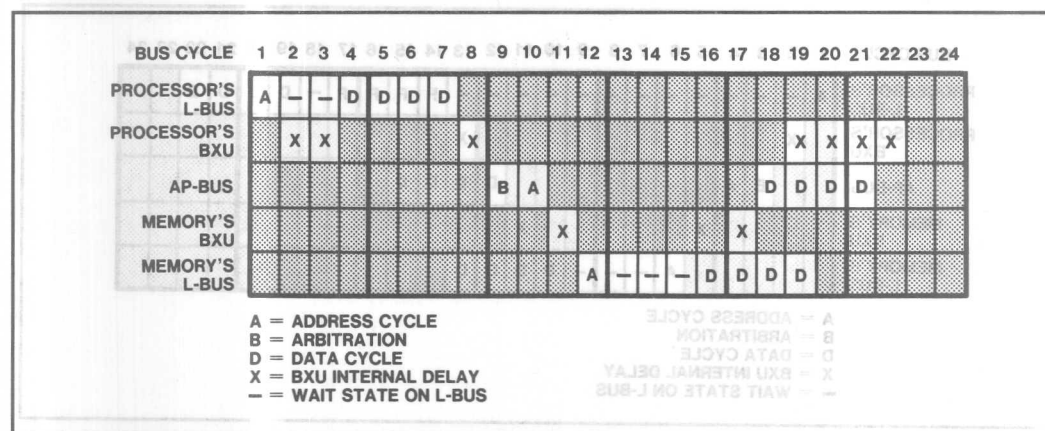


Figure 8-20: Typical I/O Prefetch Read with Buffer Fill Timing Diagram

An I/O prefetch channel in a dual AP-bus system (assuming that memory is interleaved across the two buses) can transfer data at a rate of 50-Mbytes every second. The increase in bandwidth is made possible because the prefetch unit accesses four buffers instead of the two buffers. The maximum L-bus bandwidth (assuming back-to-back 16-byte read operations with two wait states) is 32-Mbytes every second.

Another way to look at the performance of the prefetch unit is to examine the prefetch unit's ability to isolate L-bus prefetch reads from the potentially long latencies that could occur on the AP-bus in multiprocessor configurations. Normally, one can expect approximately 15 wait states for a reply to an AP-bus request. The prefetch mechanism in a two bus system allows a sustainable 10-Mbyte transfer rate if the AP-bus latency is 100 wait states.

SYSTEM CONFIGURATIONS

A multiprocessor 80960MC based system is composed of a set of modules connected to system AP-buses. Figure 8-21 shows three types of modules: active, passive, and the combination of an active and passive (active/passive). These three types of modules can be used to build a multiprocessor system.

The active and active/passive modules in a system are connected to all system AP-buses. Passive modules may be connected to a subset of the system buses. For example, if the system design used

four AP-buses, the active and active/passive modules would be connected to all four AP-buses; whereas, the passive module may connect to any or all of the AP-buses. Guidelines for the configurations are shown below:

- Each module can be connected to as many as four system AP-buses. Each system bus connection requires an individual BXU.
- Each active module can support up to two 80960MC processors (limited by the BXU's support for IAC messages). The number of other components is only limited by the electrical and physical constraints of the modules L-bus implementation.
- Logical addressing allows up to 32 modules for every system, although electrical considerations typically limit the practical number of modules to 20 per system.

The following sections describe the details of each module.

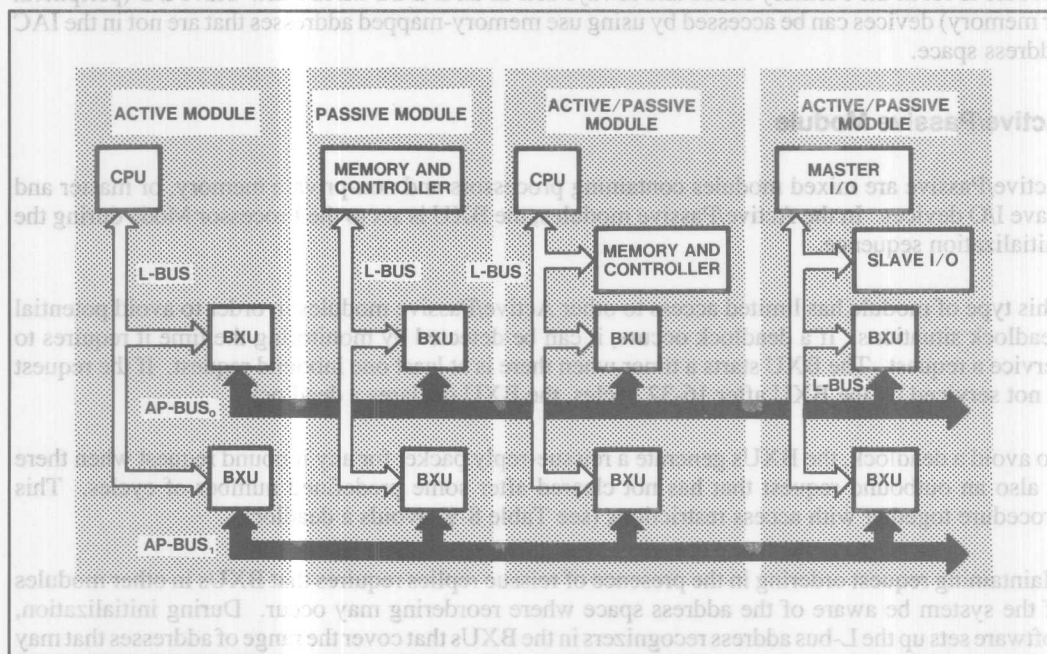


Figure 8-21: Types of Modules

Active Module

In the active modules, the L-bus traffic flows either on the L-bus to its resources (cache, etc.) or flows through the BXU on to the AP-bus. No request packets flow from the AP-bus through the BXU to the L-bus (all requests originate on the L-bus).

The BXU acts as an agent for the 80960MC processors and handles all IAC transactions on behalf of the processor. The BXU informs the 80960MC processor of an incoming IAC message by asserting the $\overline{\text{IAC}}$ interrupt pin of the 80960MC processor.

In an active module, the BXU is set to Processor mode during the initialization sequence (see *LBI Control Register* in Appendix A). When operating in this mode, the BXU is always an L-BUS SLAVE and consequently, does not need to arbitrate for control of the L-bus.

Passive Module

Passive modules are memory modules or modules with only slave I/O devices. In these modules, all L-bus requests originate from the 80960MC processor of an active module. Bus traffic flows from the 80960MC processor, through a BXU attached to the active module, onto the AP-bus, through another BXU attached to the passive module. For this case, the BXU that is attached to the passive module is set in the Memory Mode and always acts as an L-BUS MASTER. Slave I/O (peripheral or memory) devices can be accessed by using memory-mapped addresses that are not in the IAC address space.

Active/Passive Module

Active/Passive are mixed modules containing processors and non-private memory, or master and slave I/O devices. In the Active/Passive modules, the BXU is set to the Processor Mode during the initialization sequence.

This type of module has limited access to other Active/Passive modules in order to avoid potential deadlock situations. If a deadlock occurs, it can be detected by monitoring the time it requires to service a request. The BXU starts a timer when there is at least one inbound request. If the request is not serviced by the BXU after 16-32 cycles, the BXU declares a deadlock.

To avoid a deadlock, the BXUs generate a reissue-reply packet for any inbound request when there is also an outbound request that has not cleared after some predefined number of cycles. This procedure together with access restrictions (see Table 8-9) avoids a deadlock.

Maintaining request ordering in the presence of reissue replies requires that BXUs in other modules of the system be aware of the address space where reordering may occur. During initialization, software sets up the L-bus address recognizers in the BXUs that cover the range of addresses that may send reissue replies. When a request is received by a BXU that is assigned one of these special address ranges (i.e., the *Sequence* bit in the *Match* register is set), the BXU inserts wait states on the L-bus until the request is completed. This guarantees correct ordering of requests (multiple requests are not allowed). Note that ordering is guaranteed by the requesting BXU, not the BXU in the active/passive module.

Table 8-10 shows the various communication paths that are allowed between module types. A module cannot send information to a module of its own type or a type that can send information to it. These restrictions allow the reissue reply to break a deadlock.

Table 8-10: Access Restrictions

Source Module	Destination Module		
	Active	Passive	Active & Passive
Active	BXU	Yes	Yes*
Passive	—	—	—
Active/ Passive	BXU	Yes	BXU

NOTES:

BXU — Access restricted to IACs (captured by BXU).

Yes — Memory accesses and IACs allowed.

Yes* — Special setup in requesting BXUs to maintain access order

There is a possibility that bus starvation can occur, however. To avoid starvation conditions and the performance degradation of reissue replies, the system configuration and software usage of the configuration should be structured to avoid frequent occurrence of these access conflicts.

SUMMARY

The BXU component provides the interface between the L-bus and AP-bus and makes possible high performance, modular system configurations. It supports a L-bus cache memory, IAC message passing, I/O prefetching, memory, and fault tolerance. These functions are generated by the seven logic blocks listed below:

- AP-bus Interface
- L-Bus Interface
- Cache Directory and Control Logic
- Memory Support Logic
- IAC Support Logic
- I/O Prefetch Logic
- Fault Tolerance Support (described in Part III of this manual)

The BXU provides two modes of operation: Processor mode or Memory mode. In Processor mode, the BXU acts either as a L-bus master or slave, and supports the cache, prefetch, and IAC message functions. In Memory mode, the BXU acts as a L-bus master and provides support for memory functions.

To provide a flexible interface to the systems software, the BXU has various programmable registers. In addition, the BXU contains diagnostics for testing portions of the Cache Directory and Control Logic.

Memory and I/O Interface Using the BXU

9

CHAPTER 9 MEMORY AND I/O INTERFACE USING THE BXU

The M82965 BXU provides many features that enhance high-performance multiprocessor designs. This chapter outlines approaches to design memory and I/O systems in a multiprocessor design that uses the BXU as the interface between the L-bus and AP-bus. The example design uses passive and active/passive modules, which allow systems to be built and expanded in modularity. Because these modules contain a L-bus, the memory and I/O interfaces previously described in Chapters 4 and 5 are directly applicable.

BASIC MEMORY INTERFACE

In a multiprocessor design, the 80960MC processor communicates with the system memory over the AP-bus. As shown in Figure 9-1 the active module, which contains the 80960MC processor, transmits or receives data to or from the passive module, which contains the system memory and controller. The BXU that interfaces to the system memory and controller is set to Memory mode and acts as a bus master. Because the L-bus is used, the memory controller discussed in Chapter 4 is directly applicable.

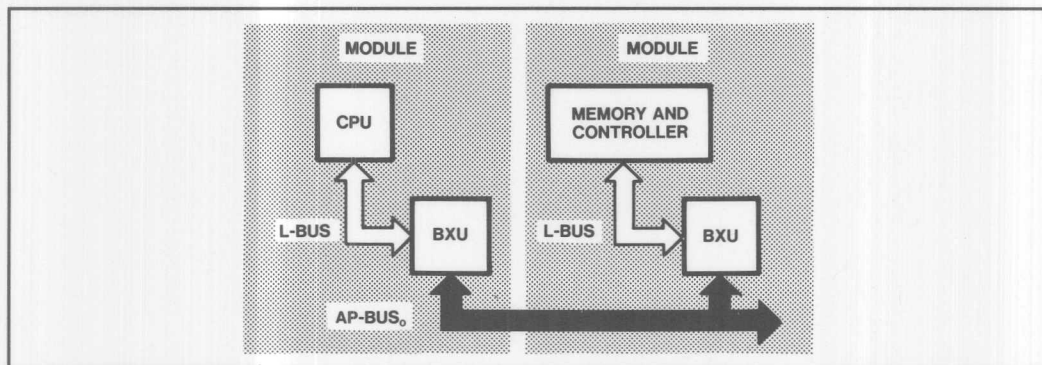


Figure 9-1: Memory Interface Logic to the BXU

Figure 9-2 shows the major logic blocks of the memory interface circuit. The data transceivers buffer the data to compensate for any slow devices that may be connected to the BXU. The address latches demultiplex the address/data signals from the BXU and latch the address. The address decoder selects the appropriate memory device from the latched address. To accommodate a memory burst transaction, the burst logic decrements the word count, increments LAD_3 and LAD_2 , and generates a CYCLE-IN-PROGRESS signal. The timing control generates a READY signal and other specific signals required by a SRAM. The byte enable latch stores the byte enable signals.

The DRAM controller multiplexes the address into a row and column address, performs the refresh operation, arbitrates between a refresh request and memory request, and generates the necessary control signals for the DRAM. The SRAM interface conditions the control signals for the SRAM.

This memory interface is identical to the one discussed in Chapter 4 because the BXU generates the L-bus signals. See Chapter 4 for details concerning the description and timing of the major logic blocks.

I/O INTERFACE

In a multiprocessor design, the 80960MC processor also communicates with the peripheral devices over the AP-bus. As shown in Figure 9-3 the active module, which contains the 80960MC processor, transmits or receives data to or from the passive module, which contains slave-type I/O devices, or an active/passive module, which contains a master I/O device and slave I/O device.

The BXU that interfaces to the passive module is set to Memory mode and acts as a bus master. Because the L-bus is used, the general I/O interface circuit discussed in Chapter 5 is directly applicable.

The BXU that interfaces to an active/passive module acts as both a L-bus master and slave. When the BXU acts as a bus slave, the module must contain logic that can issue commands to the BXU. This logic can be comprised of discrete circuits or it can be a 80960MC processor.

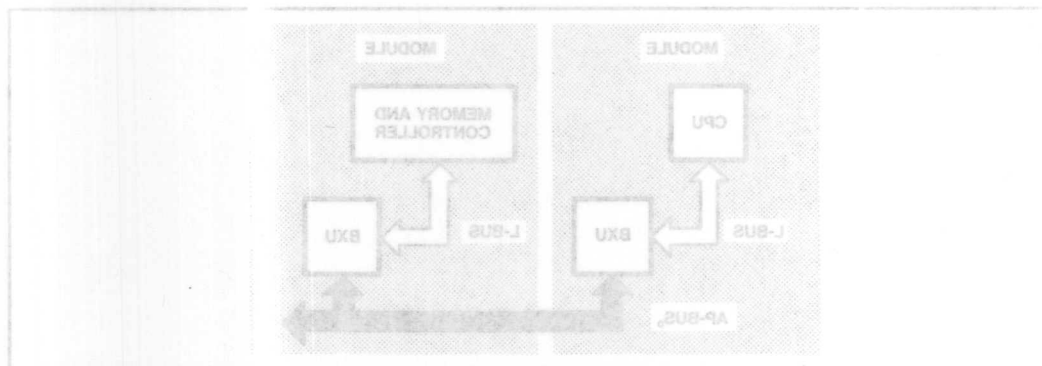


Figure 9-1: Memory Interface Logic to the BXU

Figure 9-2 shows the major logic blocks of the memory interface circuit. The data transceivers buffer the data to compensate for any slow devices that may be connected to the L-bus. The address latches the address/data signals from the BXU and latch the address. The address decoder selects the appropriate memory device from the latched address. To accommodate a memory burst transaction, the burst logic decrements the word count, increments LAD, and generates a CYCLE-IN-PROGRESS signal. The timing control generates a READ signal and other specific signals required by a SRAM. The byte enable latch stores the byte enable signals.

The DRAM controller multiplexes the address into a row and column address, performs the refresh operation, arbitrates between a refresh request and memory request, and generates the necessary control signals for the DRAM. The SRAM interface conditions the control signals for the SRAM.

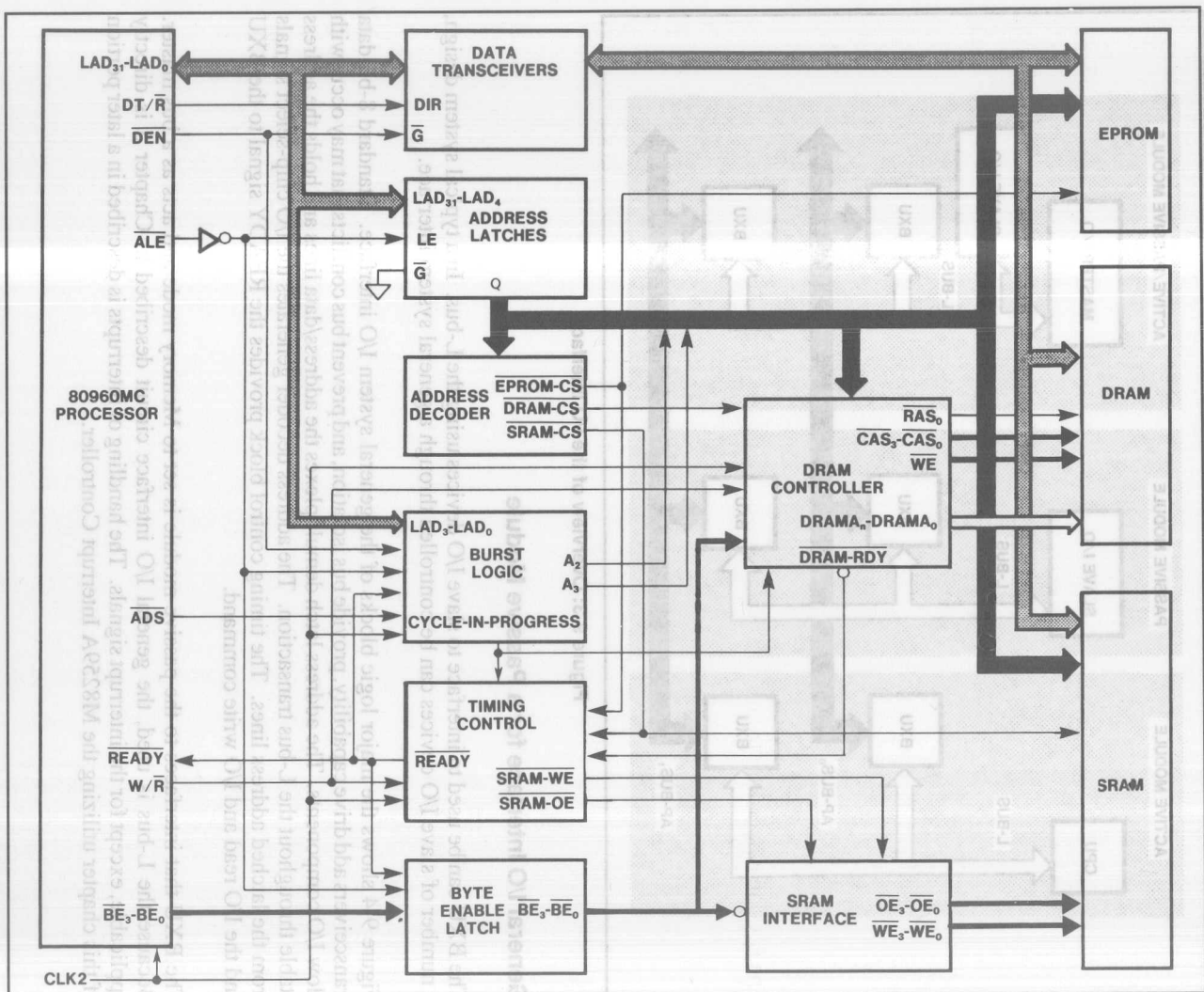


Figure 9-2: Simplified Block Diagram for Memory Interface Logic to the BXU

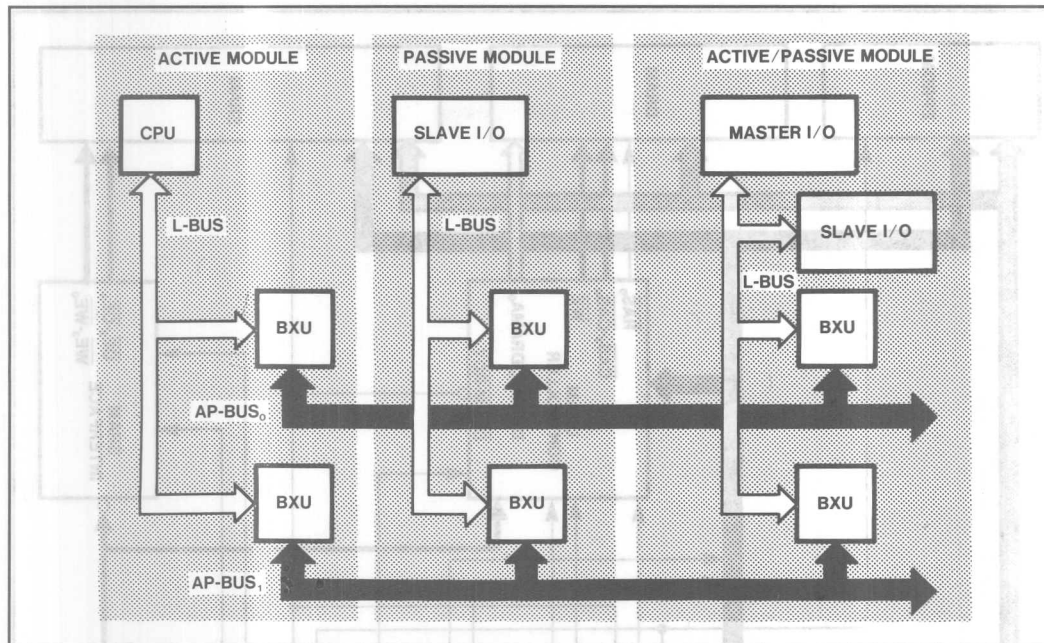


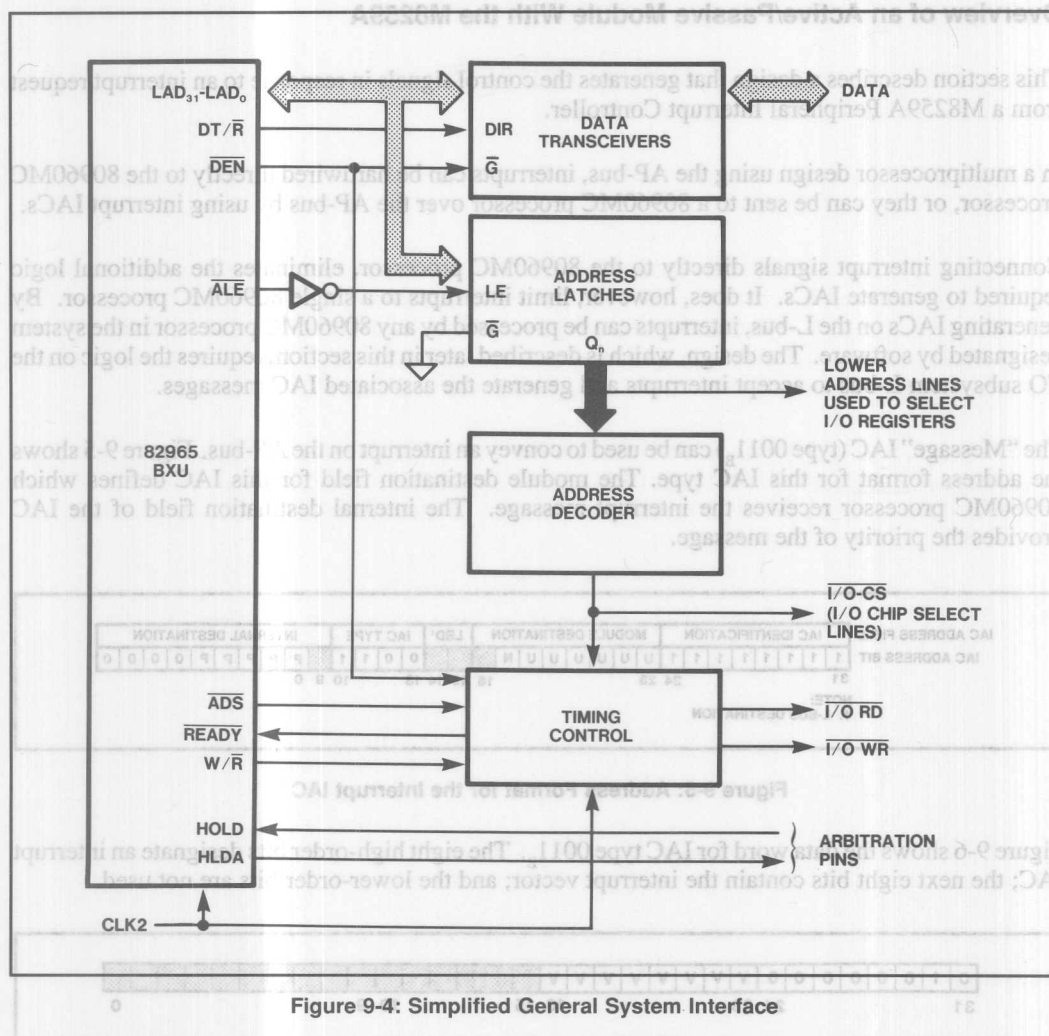
Figure 9-3: Overview of the I/O Interface

General I/O Interface for a Passive Module

The BXU can be used to interface to slave I/O devices using the L-bus. In a typical system design, a number of slave I/O devices can be controlled through a general system interface.

Figure 9-4 shows the major logic blocks of the general system I/O interface. Standard 8-bit data transceivers add drive capability, provide bus isolation, and prevent bus conflicts that may occur with slow I/O components. The address latch demultiplexes the address/data lines and holds the address stable throughout the L-bus transaction. The address decoder generates the I/O chip-select signals from the latched address lines. The timing control block provides the READY signal to the BXU and the I/O read and I/O write command.

The BXU that interfaces to the passive module is set to Memory mode and acts as a bus master. Because the L-bus is used, the general I/O interface circuit described in Chapter 5 is directly applicable, except for the interrupt signals. The handling of interrupts is described in a later portion of this chapter utilizing the M8259A Interrupt Controller.



I/O Interface for an Active/Passive Module

The L-bus signals must be generated for an active/passive module when data is sent to another active module containing a 80960MC processor. L-bus signal generation can be derived in several ways depending on the type of peripheral I/O device that is used.

This function can be performed by a 80960MC processor. However, if the computational power of a 80960MC processor is not needed, a simple controller comprised of standard logic circuits can be designed to generate the specific control signals required for L-bus operation.

Overview of an Active/Passive Module With the M8259A

This section describes a design that generates the control signals in response to an interrupt request from a M8259A Peripheral Interrupt Controller.

In a multiprocessor design using the AP-bus, interrupts can be hardwired directly to the 80960MC processor, or they can be sent to a 80960MC processor over the AP-bus by using interrupt IACs.

Connecting interrupt signals directly to the 80960MC processor, eliminates the additional logic required to generate IACs. It does, however, limit interrupts to a single 80960MC processor. By generating IACs on the L-bus, interrupts can be processed by any 80960MC processor in the system designated by software. The design, which is described later in this section, requires the logic on the I/O subsystem L-bus to accept interrupts and generate the associated IAC messages.

The "Message" IAC (type 0011_B) can be used to convey an interrupt on the AP-bus. Figure 9-5 shows the address format for this IAC type. The module destination field for this IAC defines which 80960MC processor receives the interrupt message. The internal destination field of the IAC provides the priority of the message.



Figure 9-5: Address Format for the Interrupt IAC

Figure 9-6 shows the data word for IAC type 0011_B. The eight high-order bits designate an interrupt IAC; the next eight bits contain the interrupt vector; and the lower-order bits are not used.

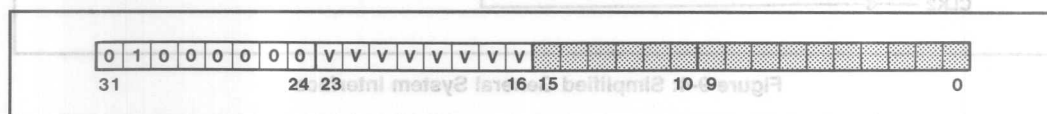


Figure 9-6: Data Word for the Interrupt IAC

When a BXU receives an IAC destined for its 80960MC processor, the BXU checks the message priority (the value in the internal destination field). If the priority value is 31, or greater than the priority of the currently executing process, then the 80960MC processor's IAC pin is asserted and the IAC message is read. If the priority is less than that of the current process, or if the BXU's IAC message buffers are full, then the BXU sends a no-acknowledgement reply packet on the AP-bus.

When the interrupt IAC is accepted by the 80960MC processor, the implicit priority of the interrupt vector is also compared with the priority of the currently executing process. If it is higher than the

current process or 31, then the interrupt is serviced. If not, then the interrupt is posted to be serviced later.

The IAC Generator

The IAC Generator is a block of logic that interconnects an interrupt requesting source to the L-bus. The IAC Generator accepts interrupt requests from an external interrupt controller, such as the M8259A Peripheral Interrupt Controller and produces an appropriate interrupt IAC message. The BXU subsequently transmits the IAC message over the AP-bus to its ultimate destination. The IAC generator performs four distinct operations:

- It generates an interrupt IAC, which is a one-word write operation to the local BXU. To determine whether the interrupt IAC is accepted, the IAC Generator monitors the BADAC signal, and resends the interrupt IAC if it is not accepted.
- It performs an interrupt acknowledge sequence by generating two INTA pulses. The two pulses are used to fetch the interrupt vector after receiving an interrupt request from the M8259A.
- It allows 80960MC processors to address the internal registers of the M8259A for initialization and programming.
- It allows 80960MC processors to read and write to the IAC message storage area. Thus, IACs can be specified by software.

To perform the first function, the IAC Generator acts as an L-bus master. It initiates transfers and allows the BXU to control the data rate with the $\overline{\text{READY}}$ signal. The IAC Generator acts as an L-bus slave to implement the last two functions. Consequently, the IAC Generator and BXU arbitrate for the L-bus by using the HOLD and HLDA signals.

Interconnection

Figure 9-7 shows the minimum set of signals required to interface an IAC Generator to the BXU. Since the IAC generator is synchronous, $\overline{\text{ADS}}$ can replace $\overline{\text{ALE}}$ to keep the number of signals to a minimum. The $\overline{\text{BE}}_3\text{--}\overline{\text{BE}}_0$ signals are not used by the IAC Generator if the design assumes that all transfers to/from the BXU are four-byte operations. This is a reasonable assumption because non-IAC transfers usually only initialize the IAC Generator and Peripheral Interrupt Controller.

The IAC Generator uses two signals, RESET and SELECT ($\overline{\text{SEL}}$), for overall control. The RESET signal is used to force the IAC Generator to a known state and to disable interrupts. Thus, the IAC Generator does not produce AP-bus traffic prior to initialization. The $\overline{\text{SEL}}$ signal can be used to distinguish the operation for the IAC Generator from other L-bus operations.

The IAC Generator uses the standard interface to the M8259A, except for the INT signal. This signal is passed through a synchronization stage to avoid a metastable state.

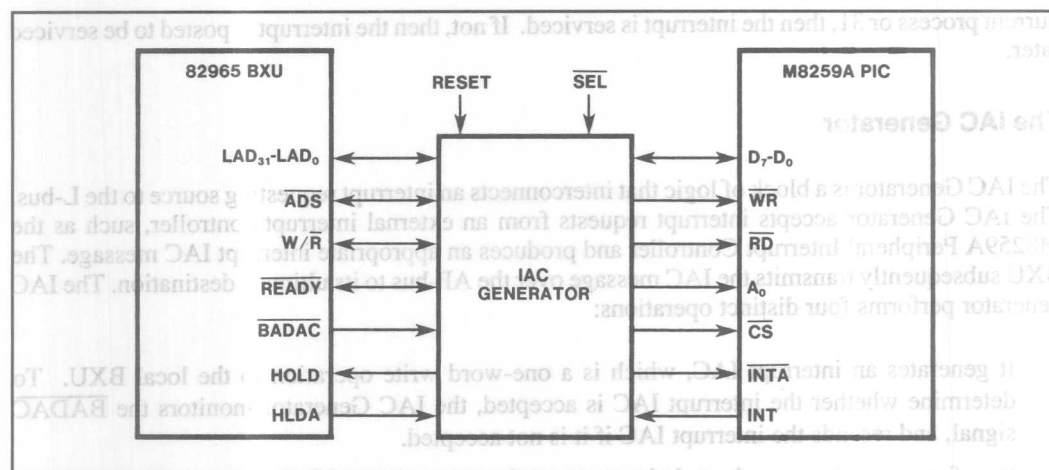


Figure 9-7: Block Diagram for Interface Circuit

Timing

Figure 9-8 shows the signal timing for the interrupt IAC messages. After the IAC Generator acquires the L-bus through the HOLD/HLDA protocol, it asserts the $\overline{\text{ADS}}$ signal and drives an address onto the LAD lines. After one bus clock (CLK), it drives the data word of the IAC message on the LAD lines. The BXU withholds $\overline{\text{READY}}$ until a write-acknowledgement reply is received on the AP-bus. If a no-acknowledgement reply is received, then $\overline{\text{BADAC}}$ is asserted on the next cycle.

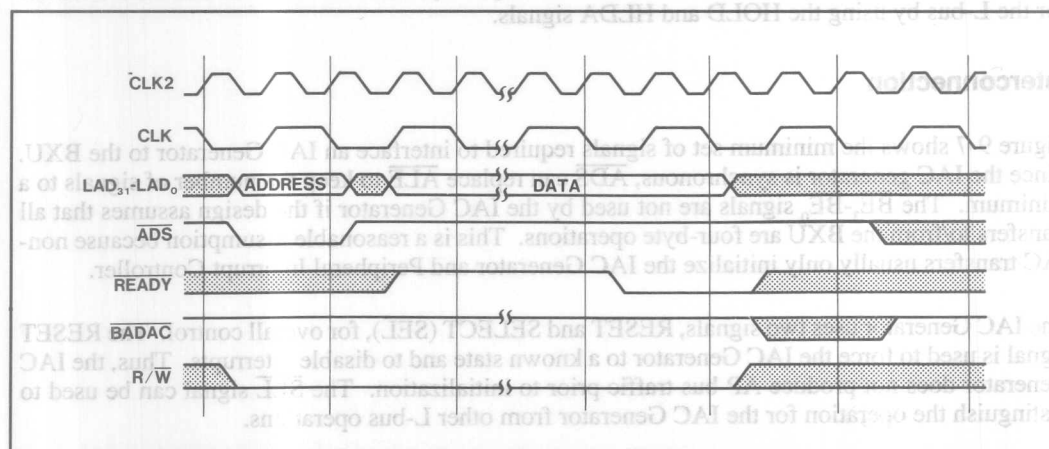


Figure 9-8: Timing Diagram for a Single-Word IAC Message

The BXU uses three-quarter cycle timing, similar to that used on the AP-bus, when it acts as a bus slave. For a write operation, the address or data is driven on the LAD lines by the bus master at the beginning of a bus clock (the A edge) and sampled by the BXU at the end of the bus cycle (the D edge, three-fourths of the way through the cycle in which $\overline{\text{READY}}$ is returned). For a read operation, the BXU drives the data on the B edge, and expects it to be sampled on the following A edge (assuming the data is ready). This signaling protocol allocates one fourth of a bus clock cycle for hold time and clock skew.

After the IAC Generator receives an interrupt request, it sends two INTA signals to the M8259A. Upon receiving the second INTA the M8259A drives an interrupt vector onto its data pins. The IAC generator latches this vector and uses it to select the appropriate IAC message (i.e., use the vector as an index into IAC message memory). The bus timing of the M8259A is slow compared to the timing of the BXU. Thus, a state machine running at the same bus clock frequency as the BXU inserts delays to properly interface to the M8259A signals.

IAC Generator Design

The IAC Generator design is shown in Figure 9-9. It consists of six logic blocks: the RAM array, state machine, interface logic, counter, address incrementer and latch, and address decoder.

The IAC messages are stored in a 32-bit wide RAM array. The depth of the RAM is a function of the number of unique messages to be stored. Normally, two words are stored for each interrupt IAC. The state machine sequences the control signals, and the interface logic generates the control signals that are routed to the BXU and the M8259A. The counter provides the necessary signal delays for interfacing to the M8259A.

The address incrementer and latch performs multiple functions: during interrupt acknowledge sequences, it latches the vector supplied by the M8259A; during IAC messages, it supplies the address to the IAC message RAM and increments this address for successive word accesses. In addition, this logic latches the address supplied by the BXU during IAC message memory accesses, M8259A register accesses, and memory-mapped commands.

The address decoder generates the chip select signals for the RAM and M8259A and decodes memory-mapped commands received from the 80960MC processors. The 80960MC processor uses two memory-mapped commands: a mask interrupt request, and an unmask interrupt request. The mask interrupt request is used to ensure that interrupts are not generated during initialization. During RESET, the interrupt requests must not be sent to prevent spurious IAC messages prior to BXU initialization. The mask interrupt request command also allows testing of the unmask command and enables the IAC Generator to write to RAM. The unmask interrupt command enables interrupt IACs after system initialization.

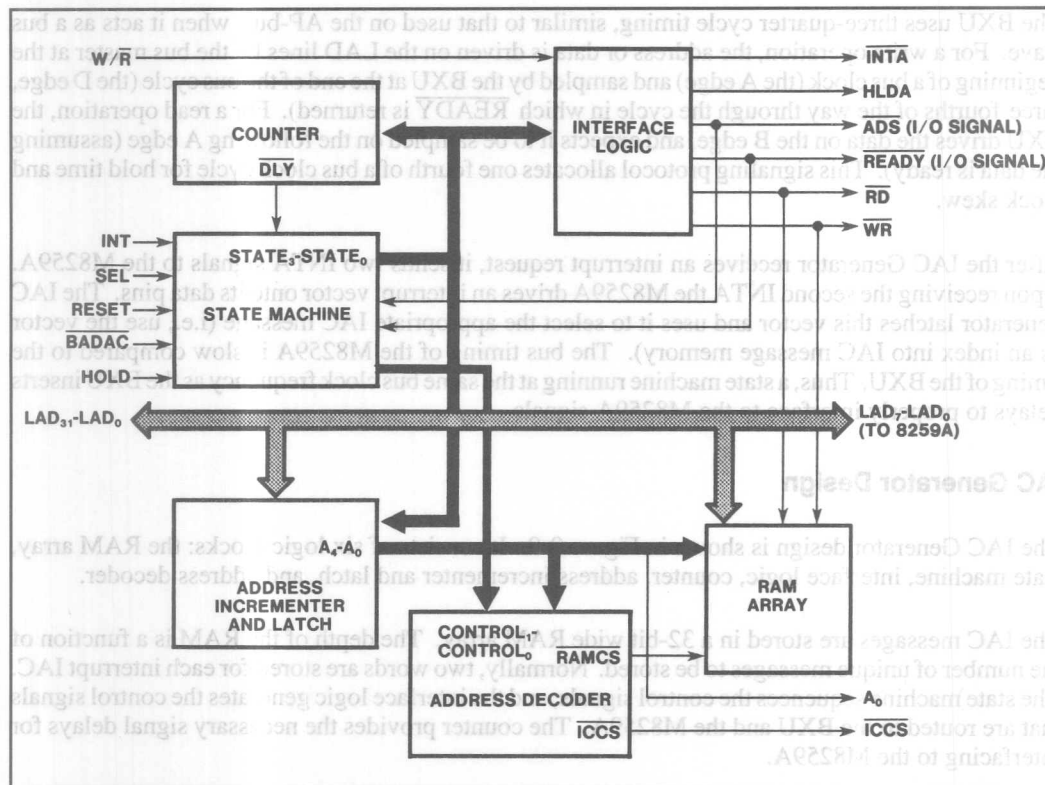


Figure 9-9: Block diagram for IAC Generator

State Diagram

Figure 9-10 shows a state transition diagram for the state machine. Table 9-1 defines each state.

This design assumes the following conditions:

- The **RESET** signal forces a transition to the idle state (I).
- The state machine is the primary bus master (other devices, including the **BXU**, must assert **HOLDR** to acquire the bus).
- An additional bit is available to store the interrupt mask bit.

State	Name	Signals Asserted	Other Action	Comments
I	Idle	None	Set <i>MASK</i> bit after reset	Waiting for <i>INT</i> from M8259A, or <i>HOLDR</i> from BXU
A ₀	Address ₀	HLDA	Latch LAD ₄ -LAD ₀ if <i>ADS</i> is asserted	Grant bus to BXU; place <i>ADS</i> in high impedance
A ₁	Address ₁	HLDA	Decode latched address	Wait here if <i>SEL</i> not asserted
DA ₀	Data Access ₀	HLDA, <i>RD</i> , or <i>WR</i>	Start delay timer	Wait for <i>DLY</i> to meet M8259A TRLRH spec
DA ₁	Data Access ₁	HLDA, <i>RD</i> , or <i>WR</i> , <i>READY</i>	Set or reset <i>MASK</i> bit if commanded	Latch data at end of cycle if write operation; assert <i>READY</i> to complete transfer
BW	Bus Wait	HLDA, <i>READY</i>	None	Wait for BXU to release L-Bus
IM ₀	IAC Message Address	<i>ADS</i> , <i>RD</i>	Increment IAC RAM address at end of cycle	Send IAC address; place <i>READY</i> in high impedance
IM ₁	IAC Message Data Word ₁	<i>RD</i>	Increment IAC RAM address at end of cycle	Send Data Word ₁ ; increment address Modulo 2
CK	Check for BADAC	None	None	If BADAC asserted retry IAC message
IA	Interrupt Acknowledge	INTA	None	Wait for <i>DLY</i> to meet M8259A TRLRH spec
II	Interrupt Idle Time	None	None	Wait for <i>DLY</i> to meet M8259A TRHRL spec
VF ₀	Interrupt Vector Fetch ₀	INTA	*Latch shifted vector at end of cycle	Wait for <i>DLY</i> to meet M8259A TRLRH spec
VF ₁	Interrupt Vector Fetch ₁	INTA	None	Interrupt vector decode wait

NOTE:

* Since each interrupt IAC message occupies two words in RAM, the vector must be shifted left by 1 bit in order to serve as a valid IAC message address.

Figure 9-10: State Diagram for IAC Generator

Address state 0 (A_0) and Address state 1 (A_1) allow the BXU or another bus master to arbitrate for the L-bus and check that the current access is for the IAC Generator. Data Access state 0 (DA_0) and Data Access state 1 (DA_1) generate the timing for accessing the internal registers of the M8259A or IAC message RAM. During the Bus Wait state (BW), the IAC Generator gains control of the L-bus again.

The interrupt acknowledge operation and IAC message transmission are performed by the following state sequence: the Interrupt Acknowledge state (IA), the Interrupt Idle state (II), the Vector Fetch 0 state (VF_0), the Vector Fetch 1 state (VF_1), the IAC Message 0 state (IM_0), the IAC Message 1 state (IM_1), and the Check state (CK).

The values of the signals are a function of current state except for \overline{RD} and \overline{WR} . These signals are conditioned by the W/\overline{R} output of the BXU, when it controls the L-bus.

SUMMARY

A multiprocessor system can be designed by using active, passive, and active/passive modules. Each module contains at least one BXU, which provides a L-bus to AP-bus interface.

A passive module can be used to supply the system memory. In this case, the BXU is set to Memory mode and generates the L-bus signals. The memory interface circuit described in Chapter 4 can be used.

A passive module can be used to communicate with slave I/O devices. In this case, the BXU is also set to Memory mode and generates the L-bus signals. The general interface circuit described in Chapter 5 can be used.

An active/passive module can be used to communicate with slave I/O devices and master I/O devices. In this case, the BXU acts as a bus master and a bus slave, and, consequently, arbitrates for the L-bus. When the BXU acts as a bus slave, a controller is required to send signals to the BXU, which passes them to the AP-bus. This controller can be a 80960MC processor or a circuit comprised of standard logic devices. For example, an IAC Generator designed for the M8259A, was illustrated in this chapter.

Fundamental Concepts of Fault Handling

10

CHAPTER 10 FUNDAMENTAL CONCEPTS OF FAULT HANDLING

A fault handling cycle consists of four phases: error detection, error confinement, error reporting, and recovery. During the detection and confinement phases, hardware errors and software bugs are detected and isolated to an area with a tightly defined interface. These faults are reported throughout the system during the error reporting phase. Finally, recovery mechanisms mask the effects of the fault from the rest of the system and if possible repair the faulty subsystem.

This chapter introduces a model for fault handling based upon these four different phases and shows the implementation of this model in the 80960 hardware system.

A MODEL FOR FAULT HANDLING

A system is generally made up of a hierarchy of fault handling cycles. Although the definition of the levels varies depending on system design and the technology of implementation, most current systems contain the following five levels in ascending hierarchical order.

1. **Component Level.** This is the lowest level of the hardware system. Failures occur within the component itself, such as the loss of a memory bit within a RAM chip. This failure could be detected by employing checksums across the internal rows of storage cells in the chip. Recovery might be implemented by an on-board associative memory, which provides back-up storage for bad cells in the normal array. By employing fault handling strategies, such as checksum and back-up memory, higher levels in the system may be totally unaware that a fault occurred and was corrected.
2. **Module Level.** This level is composed of a group of components (such as a 80960MC processor and BXU). The failures at this level occur in the components or the signal paths connecting the components. An example of a failure at this level would be the loss of a memory bit in a 32-bit word of a memory array (e.g., a bad solder joint at a RAM data-out pin). This failure could be detected by employing parity to cover single failures within a memory word. Recovery could be accomplished by replacing the parity bit with a single error-correcting Hamming code (ECC).
3. **Hardware System Level.** This level consists of a network of interconnected modules. Failures may occur when either a module or the interconnect network fails. An incorrect response from the memory module is an example of a failure at the hardware system level. This failure could be detected by duplicating the memory module and comparing the outputs. Recovery is possible if a redundant memory module is available to replace the faulty module.
4. **System Software Level.** The operating system is responsible for providing expanded service to the application software. Failures may occur when defects exist in the algorithms. For example, a software algorithm may allow a deadlock situation to occur. This can be detected by utilizing some secondary routines which monitor the operation of the system resources. Recovery might be accomplished by changing some of the previous resource allocation decisions.

5. **Application Level.** This level completes the system description. This software may have defects in its algorithms or specifications. The human interface of the system may even be used to detect and recover from application failures.

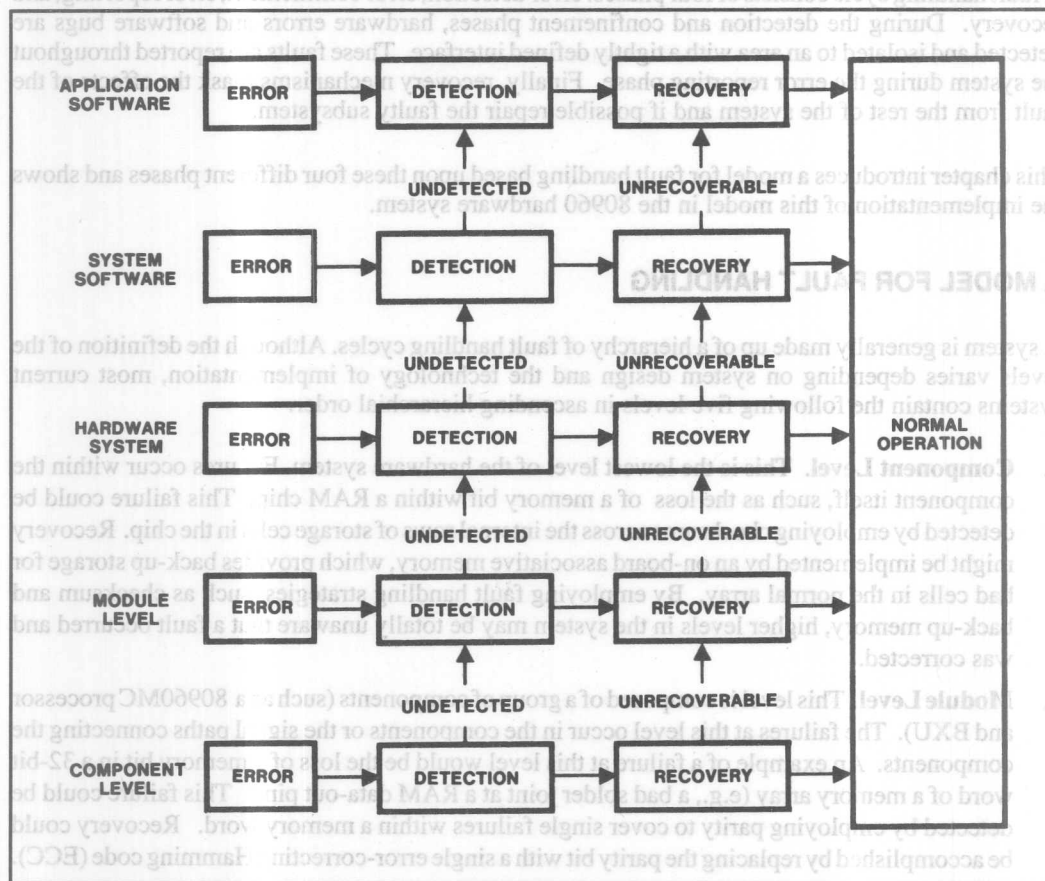


Figure 10-1: Fault Handling Model

Assuming that these system levels are connected, failures that occur at one level may flow up to higher levels if either the detection or recovery at that level is inadequate to handle the fault (see Figure 10-1). Each time a failure is reflected up, the detection mechanisms at the next level will treat the failure as if it were originally generated at the higher level. This treatment causes a loss of information about the fault by potentially masking the true cause of the failure, as well as by increasing the amount of the system which must be considered suspect.

An example of fault detection being reflected up is an undetected memory error that appears as an invalid data structure at one of the software levels. An example of recovery being reflected to higher levels is an uncorrectable memory error in an application program segment. It may be possible for

the operating system or the user to handle this fault. Recovery can also be reflected up because of improper detection or incorrect operation of the recovery mechanisms. An example of this is a 3-bit memory error that is detected as a single-bit error by the Hamming code, and is subsequently corrected.

From the preceding model, several key points about the system fault handling can be made, as listed below:

- Faults can occur at many different levels in the system.
- Each level has its own characteristics, and different detection and recovery strategies are appropriate for different levels.
- Faults not handled at one level propagate up to the higher levels in the system.
- Higher levels have more complex environments, which make recovery a more complex and slower task.

As failure modes increase in complexity, the interaction between subsystems grows, and the original source of the failure becomes more ambiguous. Thus, faults should be handled at the lowest possible level.

80960 FAULT HANDLING APPROACH

The 80960 fault handling approach provides general-purpose, adaptable, and software-transparent fault-tolerant capabilities. The 80960 system design operates under two fundamental principles. First, the number of hardware errors which are reflected up to the software levels of the system are kept to a minimum. Second, all of the fault handling mechanisms are independent and orthogonal.

Figure 10-2 shows the location of the barrier that prevents hardware failures from being reflected into higher layers of the system. It is important to note that it is impossible for any system to detect and recover from all failures that might occur. However, the 80960 hardware reduces the rate of failures reflected into the software to an extremely low level.

The reflection of errors is minimized to prevent information overload at higher levels in the system structure. If all failures are allowed to propagate to the top, the system becomes overloaded and loses its ability to react to the fault conditions. The complexity of the failure modes may make implementation impossible or force a reduction in the completeness of fault coverage.

By performing detection and recovery from hardware failures at the lowest practical level in the system, a more general and complete solution is possible. This approach divides the responsibilities for fault tolerance, allowing faster solutions to fault detection and fault recovery. The mechanisms for detection and recovery from software errors need only address the set of faults that can be generated at those levels.

By controlling and reducing the amount of errors reflected up to the next level, parallel and independent development may proceed on different levels (hardware, system software, applica-

tions). The designers at one level can safely assume that lower levels always provide consistent and correct operation.

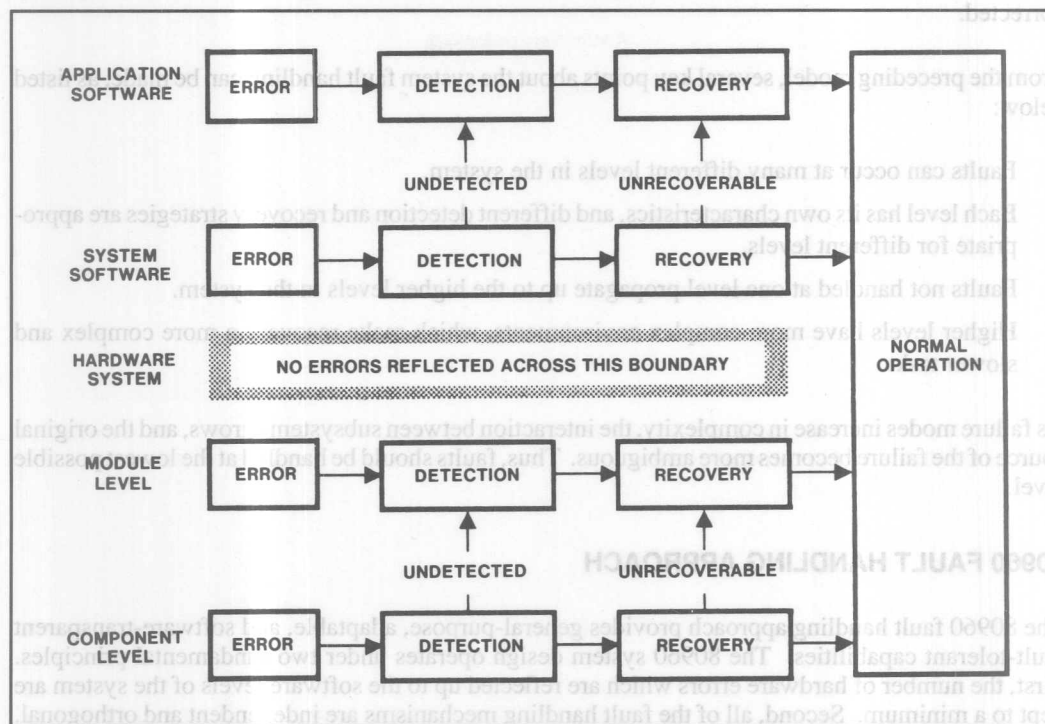


Figure 10-2: The 80960 Architecture's Separation of Hardware and Software Layers

All of the fault handling mechanisms used in the 80960 system are orthogonal. Expansion of bus bandwidth, logical resources, detection capabilities, or redundancy may be done without any side effects on the rest of the system. Minimizing the interaction between these variables provides the system with a flexible and modular basis for growth and adaptation to the application environment. System capabilities may be added or removed without affecting the application software. There is no penalty in performance, cost or system size for those fault-tolerant mechanisms not used in a system.

Although all of the hardware fault handling occurs without software assistance, software remains responsible for managing the overall fault-tolerant policy. This division of labor allows the software to tailor the resources of a system to the needs of an application without giving up any of the benefits derived from placing the fault handling mechanisms in the hardware. Software may also periodically test the detection and recovery mechanisms while the system is on-line. This allows any latent errors in the system to be uncovered before another error would force the system to face a double-failure condition.

FAULT TOLERANCE IMPLEMENTATION FOR A 80960 SYSTEM

The 80960 architecture uses VLSI replication as a basis for implementing fault-tolerant designs. By VLSI replication, an 80960 fault-tolerant system can be designed that confines and reports errors, and recovers from failures.

VLSI Replication

The replication of VLSI components is the basic principle that forms the foundation for the implementation of the 80960 fault handling mechanisms. The 80960 family offers a wide range of system configurations from a small set of VLSI components. The same components provide modular expansion of performance, memory storage, detection, and recovery capabilities. Through VLSI replication, software is only responsible for initialization, not fault detection, isolation, or recovery. There are no special-purpose components aimed solely at fault-tolerant functions.

Figure 10-3 illustrates how these components can be configured. For basic fault tolerance operation, VLSI components can be used for self-test functions, parity, and retry operations.

These components can be logically paired to form self-checking modules using a technique to detect errors called Functional Redundancy Checking (FRC). With FRC, software designates one module of the pair as the master and the other as the checker. The master generates the AP-bus requests; the checker compares all information sent to an AP-bus by the master. If either the master or the checker detect a disagreement, it will generate an error report. This is an example of module level error detection.

This master-checker pair can be replicated to form a Quad Modular Redundancy (QMR) Primary-Shadow pair. In this configuration, every FRC pair in the system is married with another self-checking FRC pair of the same type. This QMR pair of self-checking modules operates in lockstep and provides a complete and current backup for all state information in the module. QMR is also known as module shadowing because if one module (the primary) fails, a duplicate module (the shadow) is ready to operate. This QMR scheme implements in hardware the 80960 architectural models recovery mechanism.

By using BXUs, all communication in the 80960 system is done over the Advanced Processor or AP-bus. This makes orderly growth possible since no signal definition is dependent on the number of resources in the system. This approach also makes on-line repair possible. The presence or absence of any module cannot prevent communication between any other modules. The AP-bus provides a uniform and regularly structured communications path that supports the modular expansion of both fault-tolerant and standard multiprocessor system capabilities.

An 80960 system configured for fault tolerance will utilize at least two AP-Buses, and can have a maximum of four AP-Buses. This is to allow the system to recover from the failure of an AP-Bus by redirecting bus traffic over the surviving bus.

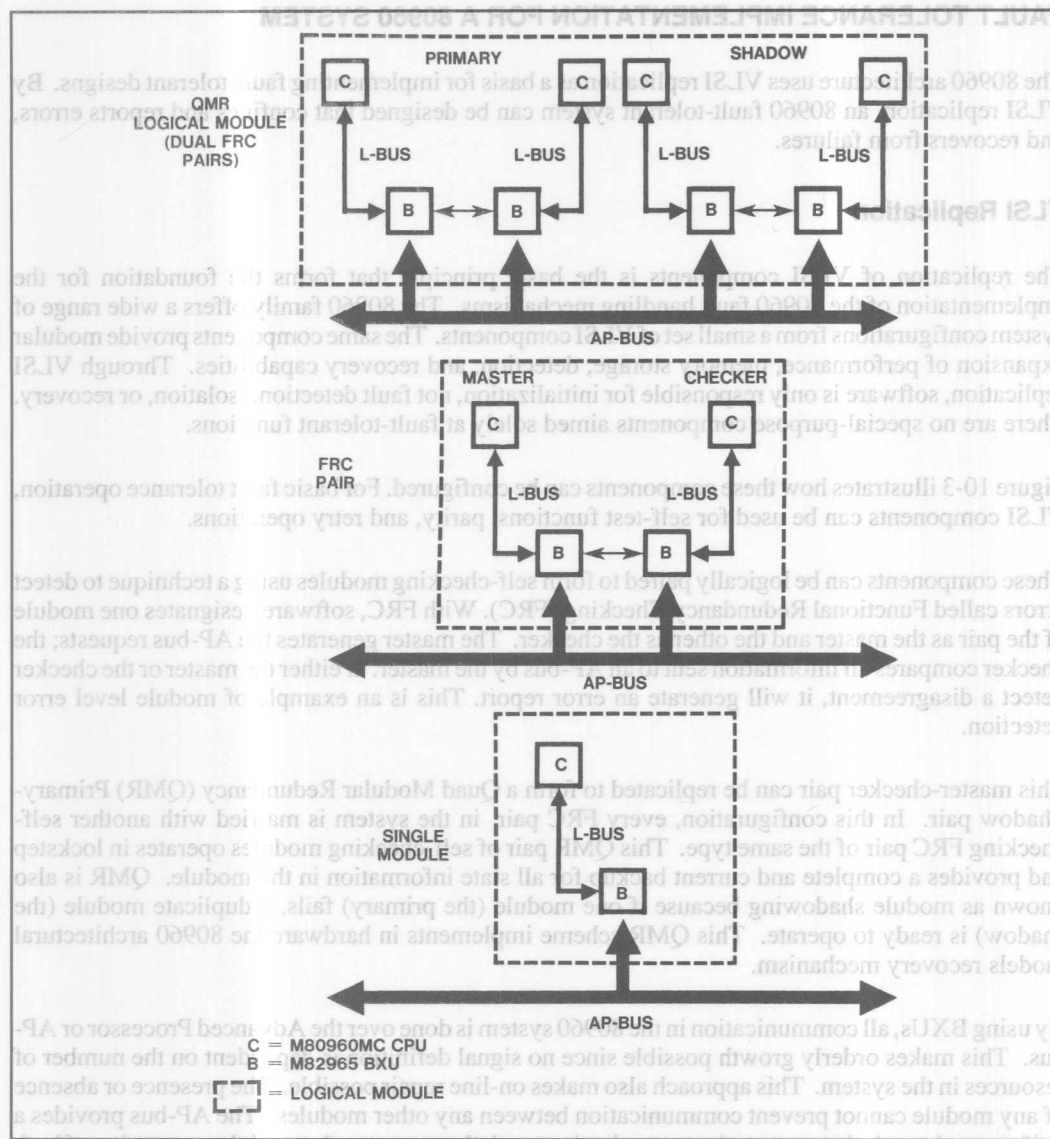


Figure 10-3: VLSI Replication

Confinement and Detection

A confinement area is defined as a unit (module or AP-bus) of the system which has a limited number of tightly controlled interfaces. The confinement area limits the damage from error propagation and localizes the faulty area for recovery and repair.

HARDWARE DESIGNER'S REFERENCE MANUAL 80960MC



FUNDAMENTAL CONCEPTS OF FAULT HANDLING



Detection mechanisms are placed at every interface to ensure that no inconsistent data can leave the area and corrupt other confinement areas. When an error occurs in the system, it is immediately isolated to a confinement area. The error is known to be in that confinement area, and all other confinement areas are known to be error-free.

By defining confinement areas, a conceptual framework is developed for the systematic and coherent placement and definition of the detection mechanisms. The confinement areas also provide a conceptual view of the system under fault conditions. This clarifies the external (software) view of the hardware and eliminates the need for diagnostic probing as a method of fault isolation.

The Reporting and Recovery Cycle

The reporting and recovery sequence for an 80960 system consists of six steps:

1. Error Reporting
2. BXU Partner Communication
3. Transient Waiting Period
4. Retry
5. Recovery
6. Probation

Figure 10-4 shows the state diagram for the error reporting cycle. After a BXU detects an error, it issues a report. When the other BXU's receive the error report, they stop their bus activity. The reporting phase lasts 256 AP-bus clock cycles from the beginning of the first report. The same reporting sequence is used independent of system configuration.

During partner communications, the paired BXUs on two separate AP-buses exchange information about the state of the outstanding requests on the failed AP-bus. This action allows the BXU on the surviving bus to correctly retry any requests that were still outstanding on the failed bus.

Errors can be of two type: **permanent** or **transient**. If a permanent error is detected, the recovery stage is started. During the recovery stage, the BXUs automatically reconfigure themselves. After recovery the BXUs enter the transient wait state. If no permanent error is detected (i.e., a transient error), the BXUs directly enter the transient wait state.

During the transient wait period, the BXUs are quiescent for 256 cycles on the AP-bus to allow system transients to settle before resuming normal system operation. This approach allows the hardware to correctly react to transient errors that are of a relatively long duration (mechanical vibration or motor startup). The transient wait period can be extended by an adjustable timer that is set by using the *Maxtime* register (see Appendix A for the description of the *Maxtime* register).

Next, the hardware system enters the retry state. During this state, all accesses that were pending at the time of the error report are retried. A BXU does not accept any new requests until it has retried

all of its previously outstanding requests. Each BXU retries its requests in the same order that it received them from the L-bus. The traffic on the AP-bus, however, may not necessarily be retried in the same order as it was before the error report.

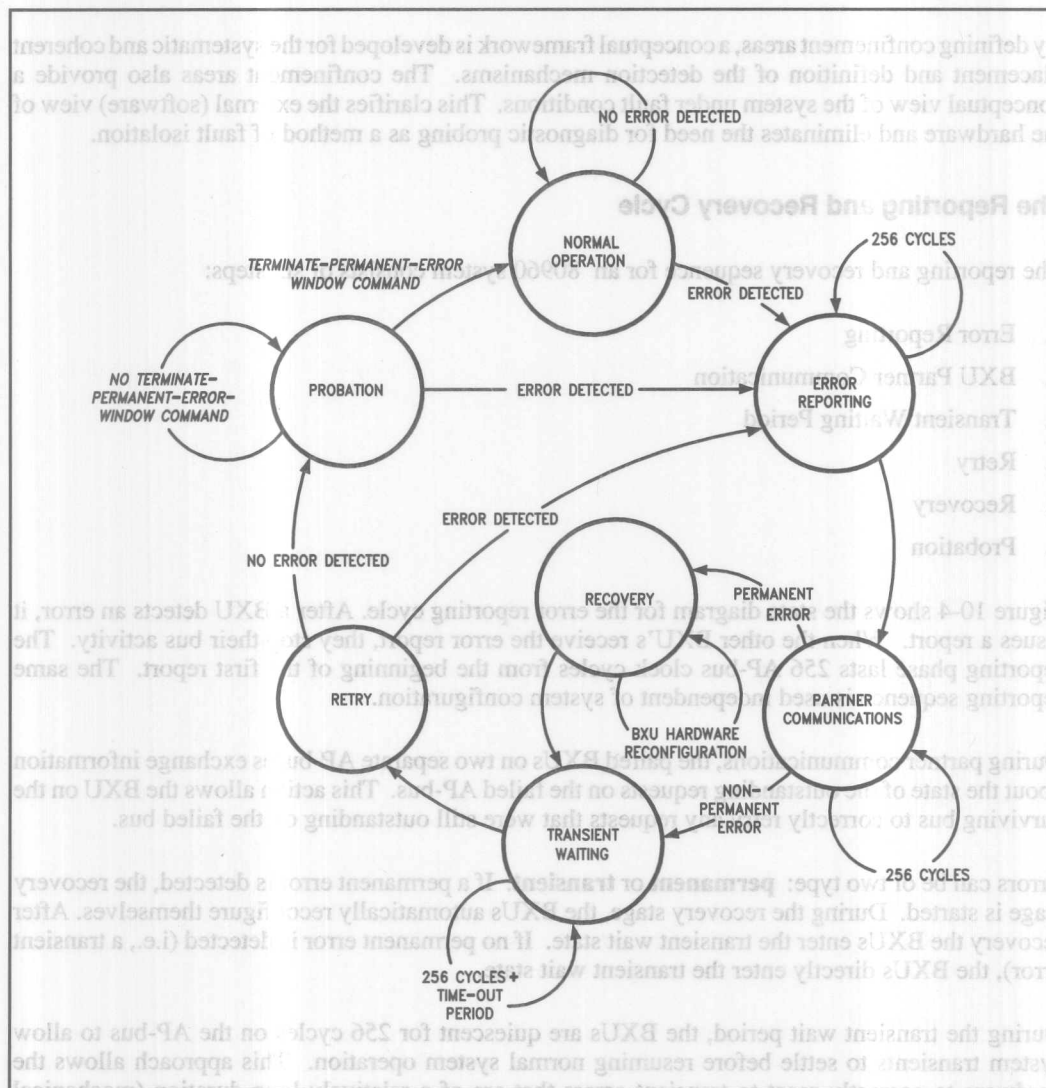


Figure 10-4: The Reporting and Recovery State Diagram

During the retry interval, the fault handling logic checks the bus activity for the occurrence of an error. If a second error is detected, a start bit is issued to begin another error reporting cycle.

A recurring transient error is considered a permanent error if it occurs again during the probation period (called the permanent error window). The time period starts when the retry state begins. The permanent error window ends when the BXU issues a *Terminate-Permanent-Error-Window* command (controlled by software) after the retry state. If a transient error occurs within this software defined time period, the BXU issues a start bit to begin the error reporting cycle. The BXU marks the error as permanent if it is the same error that previously occurred.

Fault Recovery Configuration Examples

Figure 10-5 illustrates the range of fault tolerant alternatives available to 80960 system designers. The 80960 architecture supports fault recovery through retry algorithms and reconfiguration facilities in the BXU with the replication of CPUs, BXUs, and RAM arrays. How much replication is used depends on the level of fault tolerance desired for a particular application.

Fault tolerance can range from a simplistic single module implementation utilizing software reconfiguration to a full QMR system with transparent hardware reconfiguration.

Basic Fault Recovery

Figure 10-6 shows an example of a system design that provides a rudimentary level of fault tolerance. This system has the ability to detect parity, bus arbitration, and time-out errors on the AP-bus.

Within this system, several techniques are used for error recovery. Parity errors are handled by the BXU using retry algorithms, which signal the agent that sent the faulty transaction to resend it. Time-out errors are handled by a fault handling routine executed by the processor. Recovery may not be possible if the time-out error is the result of faulty software.

Recovery from errors at this fault tolerance level is limited to transient errors. However, since transient errors account for the majority of failures in a system, this basic fault-tolerant system provides a high level of data integrity and reduces the system downtime due to transient conditions.

Errors resulting from hardware failures would require system shutdown and replacement of components if critical resources have failed.

Self-Healing and Continuous Operation Example

The previously described basic configuration is the lowest cost alternative, but it does not ensure that all errors are detected. For applications that require that all errors are detected, a self-healing system can be designed using Functional Redundancy Checking (FRC). Adding a second set of checker components to each module improves the error detection capabilities of the system, providing "high-confidence" computing. No single hardware failure will go undetected and corrupt the results of a critical computation. FRC ensures that any error is caught before it can propagate to another module in the system. FRC alone does not provide automatic hardware recovery as in a Quad Modular Redundant system, but it does detect and contain errors to ensure that the system does not become corrupted. It is then the responsibility of system software to implement a "self-healing" strategy where the faulty resource is disabled and the system reinitialized.

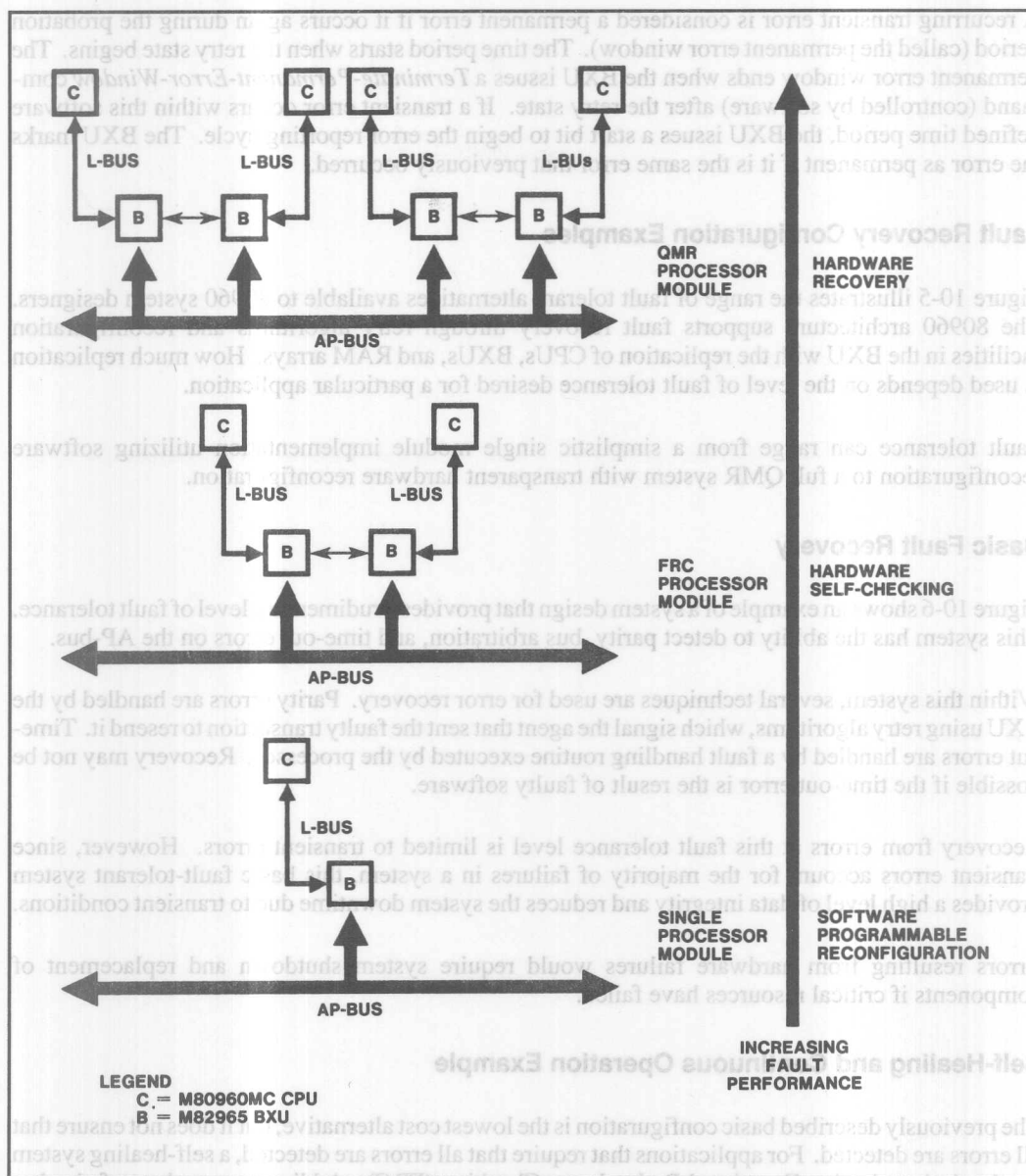


Figure 10-5: Fault-Tolerant Alternatives

In a continuous operation configuration, a redundant resource is used to replace the failed unit when a permanent error occurs. This switch is done on a BXU-to-BXU basis; there is no centralized element that controls the switch. Each BXU knows which module or AP-bus it is backing up (shadowing). If the error report identifies its partner as the faulty unit, then it becomes active and

takes over operation for the faulty unit. After the resource switch is complete, all of the outstanding accesses are retried. This allows operation to resume at a point before the failure could corrupt data.

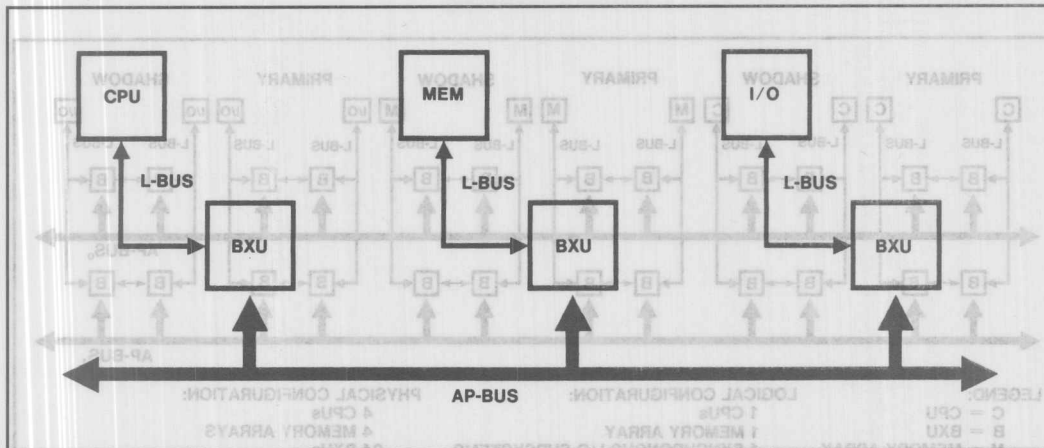


Figure 10-6: Example of a Basic Fault-Tolerant Design

These reconfiguration and recovery actions are performed by the hardware without any software intervention. After recovery is complete, system software can make policy decisions regarding the optimum system configuration, given the resources remaining in the system. These policy decisions are carried out while normal system operation continues.

Figure 10-7 shows a system capable of a comprehensive deferred maintenance. The system uses all of the available detection mechanisms, and has more than one of every resource in the system. When an error occurs, it will be detected and isolated by the hardware. Software can then rapidly reconfigure the system around the faulty module and continue operation.

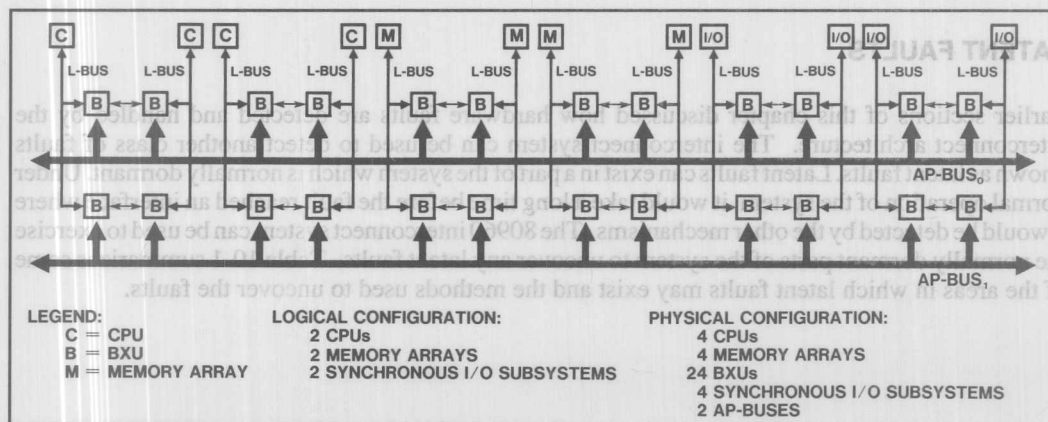


Figure 10-7: Self-Healing Multiprocessor Configuration

Figure 10-8 shows an example of a small continuous operation QMR system. This system is configured so that every resource in the system has a backup. The system can continue operation in the presence of any single failure. All of the detection and recovery mechanisms are enabled.

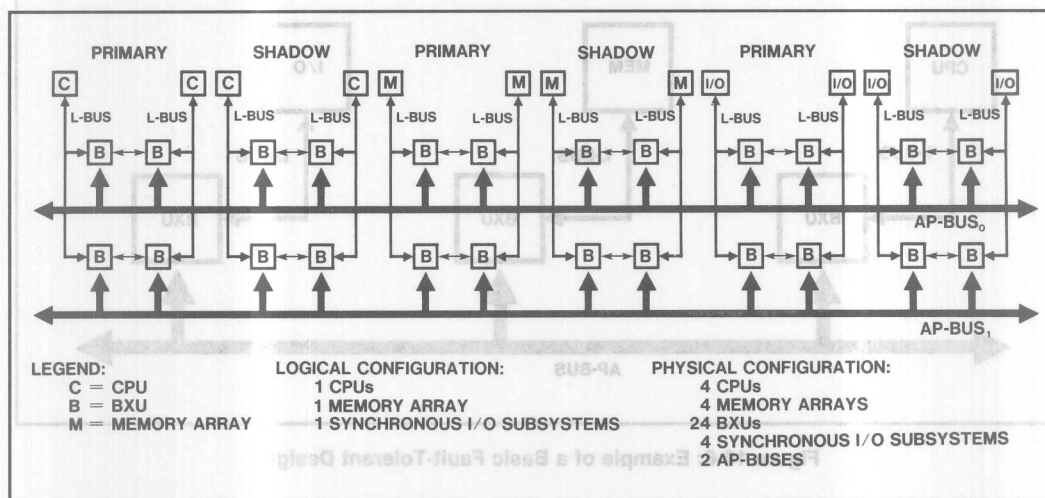


Figure 10-8: QMR Configuration

The software configurability of a BXU system allows a system to use a combination of the above strategies. In a typical naval avionics application, software can configure a system as a full QMR system for critical applications, such as an automatic carrier launch system, and then switch to an FRC-only system during flight. The software could subsequently reconfigure the system back to a full QMR system for an automated carrier landing system. This reconfiguration doubles the system throughput (twice as many processors are working in parallel) without making any hardware changes.

LATENT FAULTS

Earlier sections of this chapter discussed how hardware faults are detected and handled by the interconnect architecture. The interconnect system can be used to detect another class of faults known as latent faults. Latent faults can exist in a part of the system which is normally dormant. Under normal operation of the system, it would take a long time before the fault reached an interface where it would be detected by the other mechanisms. The 80960 interconnect system can be used to exercise the normally dormant parts of the system to uncover any latent faults. Table 10-1 summarizes some of the areas in which latent faults may exist and the methods used to uncover the faults.

Table 10-1: Exercising Latent Faults

Dormant Area	Exercise
Detection Mechanisms	Software* periodically forces error conditions into the detection mechanisms.
Reporting Mechanisms	Software* periodically initiates and observes error reports.
Recovery Mechanisms	Software* periodically invokes recovery operations.

NOTE:

* BXUs use special commands to exercise dormant areas.

SCOPE OF 80960 FAULT-TOLERANT SYSTEM DESIGN

The interconnect architecture and the VLSI components provide a stable base for developing fault-tolerant 80960 systems. The interconnect components (BXU's) address the issues concerning fault tolerance, which are encountered when constructing the 80960 central system.

A number of system-wide issues remain the responsibility of the 80960 system designer. These issues include the following:

- A fault-tolerant asynchronous I/O system
- Fault-tolerant power supplies and distribution method
- A fault-tolerant method for clock generation and distribution
- The electrical and physical provisions for on-line repair
- Environmental conditions, such as system cooling

SUMMARY

Faults can occur at various levels within the system. Handling the fault at the lowest possible level and not allowing the fault to propagate to a higher level is essential in fault-tolerant design. Higher levels have more complex environments, which make recovery a complicated and slow task. As failure modes increase in complexity, the interaction between subsystems grows, and the original source of the failure becomes more ambiguous.

The 80960 fault handling approach minimizes the hardware errors which can be reflected to higher levels. This is accomplished by duplicating VLSI components, partitioning the system into a set of confinement areas, and performing all communication over redundant buses.

The 80960 AP-Bus interconnect architecture provides a standard VLSI method for constructing multiple processor VLSI computer systems. The 80960 interconnect architecture is implemented by the 82965 BXU. Together with the 80960MC processor, these components permit the construction

of modular, extensible, multiprocessor computer systems. The components are designed to support the construction of **fully fault-tolerant 80960** systems.

The fault-tolerant mechanisms are designed to provide a flexible and complete solution to the problems of fault-tolerant hardware. For basic systems (those without FRC checkers for error detection or QMR for recovery), a user may decide to use only a few detection mechanisms and provide recovery only for transient errors. To reduce maintenance costs and increase system availability, a system may use all of the detection mechanisms (i.e., may add checker components), but may not add any extra recovery capability (i.e., may not configure self-checking modules into a fault-tolerant QMR module). Continuous operation is available to the user who adds the extra recovery capabilities.

NOTE:

* BXUS use special commands to exercise dormant areas.

SCOPE OF 80960 FAULT-TOLERANT SYSTEM DESIGN

The interconnect architecture and the VLSI components provide a stable base for developing fault-tolerant 80960 systems. The interconnect components (BXUS) address the issues concerning fault tolerance, which are encountered when constructing the 80960 central system.

A number of system-wide issues remain the responsibility of the 80960 system designer. These issues include the following:

- A fault-tolerant asynchronous I/O system
- Fault-tolerant power supplies and distribution method
- A fault-tolerant method for clock generation and distribution
- The electrical and physical provisions for on-line repair
- Environmental conditions, such as system cooling

SUMMARY

Faults can occur at various levels within the system. Handling the fault at the lowest possible level and not allowing the fault to propagate to a higher level is essential in fault-tolerant design. Higher levels have more complex environments, which make recovery a complicated and slow task. As failure modes increase in complexity, the interaction between subsystems grows, and the original source of the failure becomes more ambiguous.

The 80960 fault handling approach minimizes the hardware errors which can be reflected to higher levels. This is accomplished by duplicating VLSI components, partitioning the system into a set of confinement areas, and performing all communication over redundant buses.

The 80960 AP-Bus interconnect architecture provides a standard VLSI method for constructing multiple processor VLSI computer systems. The 80960 interconnect architecture is implemented by the 82965 BXU. Together with the 80960MC processor, these components permit the construction

*Confinement Areas/
Detection Mechanisms*

11

Containment Areas/ Detection Mechanisms

11

CHAPTER 11 CONFINEMENT AREAS/DETECTION MECHANISMS

The first step toward fault-tolerant design is to detect and isolate the error. This chapter describes the details of how the 80960 architecture defines a confinement area and the associated detection mechanisms. Specifically, this chapter explains the following items:

- AP-Bus confinement areas
- Module confinement areas
- Functional Redundancy Checking (FRC)

CONFINEMENT AREAS

A confinement area limits error propagation and localizes the faulty area for subsequent recovery and repair. Confinement areas are defined as units of the system which have a limited number of tightly controlled interfaces. Detection mechanisms are placed at every interface to ensure that no inconsistent data can leave an area and corrupt other confinement areas. As data flows from one area to another, the operation of the confinement area is rigorously checked and correct operation is guaranteed before it is passed on to the next confinement area.

Figure 11-1 illustrates the different types of confinement areas. The system bus confinement area consists of the AP-bus and part of the BXU. The module confinement area consists of the BXU and all of the components attached to its L-bus except for an asynchronous I/O port. The interface to an asynchronous I/O system is considered in Chapter 15.

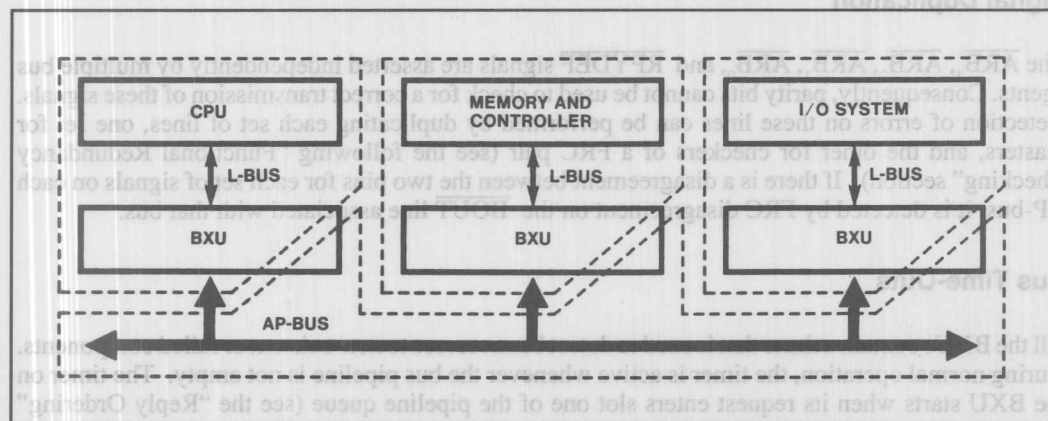


Figure 11-1: Fault Tolerance Confinement Areas

THE AP-BUS CONFINEMENT AREA

The AP-bus confinement area consists of the AP-bus and some of the interface logic in the BXU. Every BXU attached to AP-bus checks the information that flows from the AP-bus to the module. No information can leave the AP-bus confinement area and enter a module without first being checked by the BXU. To accomplish this, the BXU uses the following error detection mechanisms:

1. Two interlaced parity bits covering the $\overline{AD}_{31}-\overline{AD}_0$ and $\overline{SPEC}_5-\overline{SPEC}_0$ signals
2. Signal duplication for \overline{ARB}_3 , \overline{ARB}_2 , \overline{ARB}_1 , \overline{ARB}_0 , and \overline{RPYDEF}
3. Bus time-outs

Parity

Two interlaced parity bits (\overline{CHK}_1 and \overline{CHK}_0) are provided to check the transmission of the data, address, and specification information. Interlaced parity associates adjacent signals with different parity bits. For example, \overline{CHK}_0 is even parity across the even $\overline{AD}_{31}-\overline{AD}_0$ and $\overline{SPEC}_5-\overline{SPEC}_0$ pins; and \overline{CHK}_1 is even parity across the odd $\overline{AD}_{31}-\overline{AD}_0$ and $\overline{SPEC}_5-\overline{SPEC}_0$ pins. This alignment allows the parity detection mechanism to detect the most frequent double failure cases (shorts between adjacent lines), as well as all single bit failures. The parity bits are always valid, even when the AP-bus is idle.

The value of \overline{CHK}_1 and \overline{CHK}_0 signals are compared with the values calculated internally by the BXU. For proper timing, the receiving BXU waits 0.5 bus cycles after receiving the data before issuing a parity error.

Signal Duplication

The \overline{ARB}_3 , \overline{ARB}_2 , \overline{ARB}_1 , \overline{ARB}_0 , and \overline{RPYDEF} signals are asserted independently by multiple bus agents. Consequently, parity bits cannot be used to check for a correct transmission of these signals. Detection of errors on these lines can be performed by duplicating each set of lines, one set for masters, and the other for checkers of a FRC pair (see the following "Functional Redundancy Checking" section). If there is a disagreement between the two pins for each set of signals on each AP-bus, it is detected by FRC disagreement on the \overline{BOUT} line associated with that bus.

Bus Time-Outs

All the BXUs contain a timer that is used to detect bus accesses to non-existent or failed components. During normal operation, the timer is active whenever the bus pipeline is not empty. The timer on the BXU starts when its request enters slot one of the pipeline queue (see the "Reply Ordering" section in Chapter 7). When this BXU receives a reply or a reply deferral packet, the timer is reset.

The timer is nominally 64 bus clock cycles long. A time-out occurs if the bus is idle for more than 64 bus cycles while there is an outstanding request. When a time-out occurs, the BXU that originally sent the request issues a BAD-ACCESS reply code in the SPEC field reply packet (to the originating

CPU) to clear the request from the bus. This action prevents the 80960MC processor and the AP-bus from suspending activity because a destination does not exist.

An access may be allowed to take longer than 64 bus cycles, but the serving BXU must assert the RPYDEF signal. The timer cannot be disabled.

MODULE CONFINEMENT AREA

Except for an asynchronous I/O system, the module confinement area includes all the components that reside on the L-bus, e.g., the 80960MC processor, the associated BXUs, the support logic in the module, the RAM array, the memory controller, etc. The AP-bus is the only interface to the module confinement area. No information (control, address, or data) can leave the module confinement area without first being checked by one of the BXUs in the module. The interface to an asynchronous I/O system is considered in Chapter 15.

The module confinement area contains the BXU (except for the internal BXU logic used by the AP-bus confinement area) and all of the components attached to the L-bus of the BXU. The error detection mechanisms for the module confinement area are similar to the AP-Bus confinement checks (interlaced parity, signal duplication, bus time-outs), with the addition of Functional Redundancy Checking (FRC).

FUNCTIONAL REDUNDANCY CHECKING

Functional Redundancy Checking consists of a clock-by-clock comparison of the AP-bus pins of two BXUs. Each pair of BXUs act as a single logical unit with one operating as a master and the other as checker. These two bus agents run in lockstep, with the checker ensuring that the master is operating correctly. On every clock cycle, the checker compares its internal bus data to the data it actually observes on the AP-bus, which is driven by its master. If a disagreement is detected, then an error has occurred in either the master or checker. Also, when operating as an FRC pair, each BXU checks receipt of information from its shared local bus and transmission over the AP-Bus.

In systems that use FRC, the quality of error detection on the AP-bus is increased. FRC provides detection of errors that may occur in the interface to the AP-bus and BXU. In addition, FRC allows detection of failures in the ARB_3 , ARB_2 , ARB_1 , and ARB_0 or RPYDEF signals for a dual arbitration network.

Physical Connection

For FRC to exist, the two BXU's must be tightly coupled to form a master-checker pair. Figure 11-2 shows the interconnection of the FRC pair. Each BXU of an FRC pair must attach to the same AP-Bus, and the bi-directional Module check (MODCHK) and Buffer Out (BOUT) lines must be connected together in a point-to-point manner between the FRC pair (ie. not part of the AP-bus).

The MODCHK line is asserted by the BXU to inform its FRC twin that it has received a request from the L-bus for access to the AP-bus (MODCHK does not check whether or not the signals are correct).

The BXUs detect an FRC error when one BXU asserts $\overline{\text{MODCHK}}$ while the other does not. Both the master and the checker have the capability to detect a module checking (FRC) error. If a BXU is not asserting the $\overline{\text{MODCHK}}$ line and it senses that it is asserted (by its twin or a hardware fault), it will generate an “Unsafe Confinement Area” FRC Error report. This type of error indicates that the FRC pair has lost synchronization and is considered a permanent failure, since it cannot be determined which of the components is at fault when a disagreement occurs.

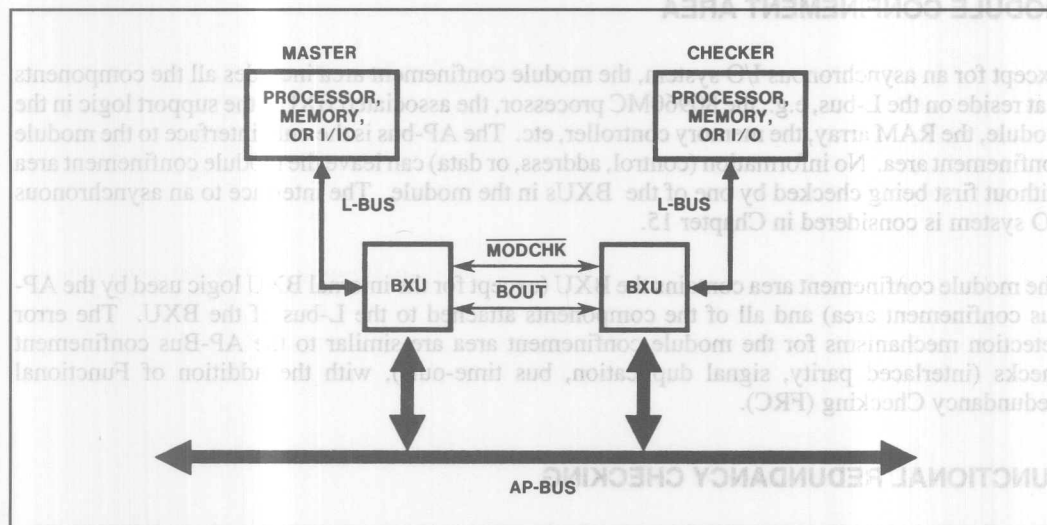


Figure 11-2: Functional Redundancy Checking

If the request for AP-Bus access has been successfully received by both BXU's without a $\overline{\text{MODCHK}}$ generated FRC error, the arbitration process for AP-Bus access begins. Both the master and checker BXU's arbitrate for the AP-Bus. After the master BXU has gained access to the AP-Bus, it drives the bus through its AP-Bus output buffers.

The Buffer Out ($\overline{\text{BOUT}}$) line is asserted by the BXU to inform its twin that it is driving the AP-Bus. Although both BXU's assert $\overline{\text{BOUT}}$, only the master actually drives the AP-Bus. The checker's output drivers are internally disabled, as it only checks the data on the AP-Bus; it never drives the bus. The $\overline{\text{BOUT}}$ line is used to detect arbitration errors by checking whether or not the BXU drives the AP-bus ($\overline{\text{BOUT}}$ does not check whether or not the signals are correct). The paired BXUs detect an FRC error when one BXU asserts $\overline{\text{BOUT}}$ while the other does not. If a BXU is not asserting the $\overline{\text{BOUT}}$ line and it senses that it is asserted (by its twin or a hardware fault), it will generate a “Bus Arbitration” FRC Error report. Errors detected by the $\overline{\text{BOUT}}$ line may be either permanent or transient since they can be induced by a hardware problem or noise on the AP-Bus.

A Bit-for-Bit comparison (exclusive or) is performed by the FRC pair on the $\overline{\text{SPEC}}_5$, $\overline{\text{SPEC}}_0$, $\overline{\text{AD}}_{31}$, $\overline{\text{AD}}_0$ lines, to check whether or not these signals are correct. FRC on the $\overline{\text{SPEC}}_5$ - $\overline{\text{SPEC}}_0$ and $\overline{\text{AD}}_{31}$ - $\overline{\text{AD}}_0$ lines provides detection for any data manipulation, data translation, or sequencing errors. The master drives the lines, and the checker verifies the validity of the data. The master also performs

a self check to verify the state reflected on it's output drivers matches the data it is driving onto the AP-Bus. Either the master or the checker can detect this type of FRC error, and will generate a "Component" FRC Error report. Functional Redundancy Checking can be disabled by clearing the *Error-Reporting-Enable* bit in the *FTI* register, or by splitting the FRC pair (see Appendix A for the description of the *FTI* register).

Functional Redundancy Checking is not performed on the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$, $\overline{\text{CHK}}_1$ - $\overline{\text{CHK}}_0$, $\overline{\text{ARB}}_3$ - $\overline{\text{ARB}}_0$, and $\overline{\text{RPYDEF}}$ lines. FRC does not apply to the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines because the error reporting protocol checks these lines. Similarly, FRC does not apply to the $\overline{\text{CHK}}_1$ - $\overline{\text{CHK}}_0$ lines because their operation is fully checked by the parity computation.

The $\overline{\text{ARB}}_3$ - $\overline{\text{ARB}}_0$ and $\overline{\text{RPYDEF}}$ pins can be configured in two ways: in a dual arbitration network, or a single arbitration network. In dual arbitration network systems, the $\overline{\text{ARB}}$ and $\overline{\text{RPYDEF}}$ signals are duplicated; one set connects to the masters, the other connects to the checkers, as shown in Figure 11-3. The duplication of these signals forms two parallel networks so that an arbitration error can be detected as an FRC error on the $\overline{\text{BOUT}}$ line. This configuration allows the detection and containment of arbitration errors before they can corrupt the system bus, but does not allow the FRC pair to be split.

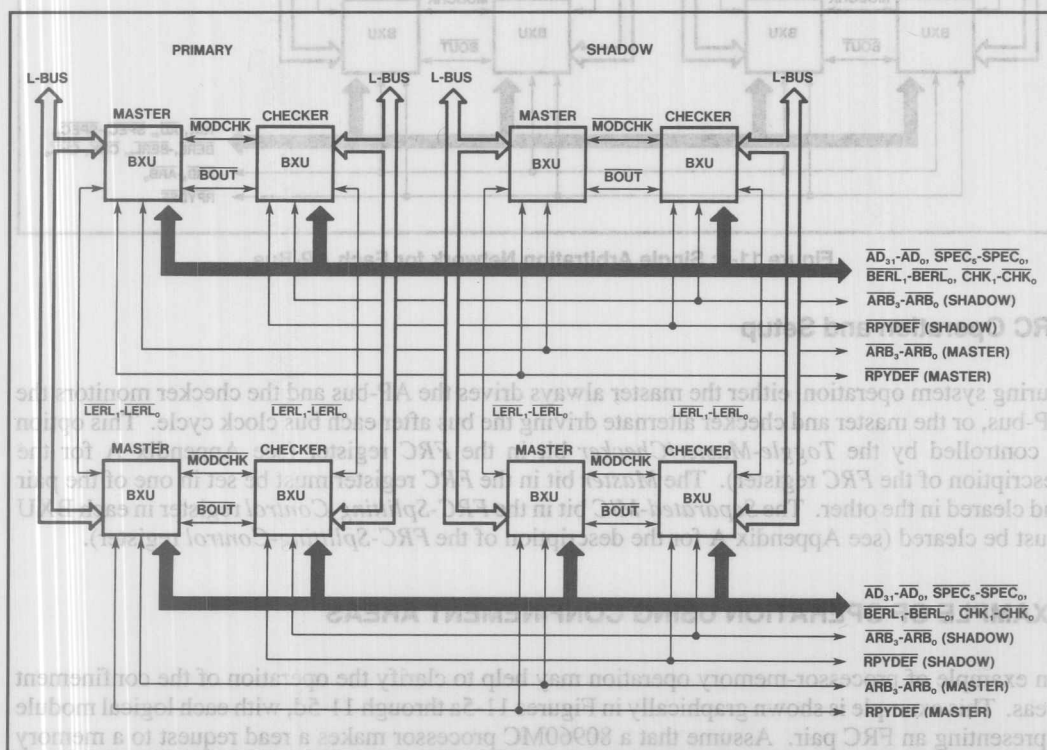


Figure 11-3: Dual Arbitration Network for Each AP-Bus

information passes through the BXU and flows across the AP-bus and to the addressed memory module.

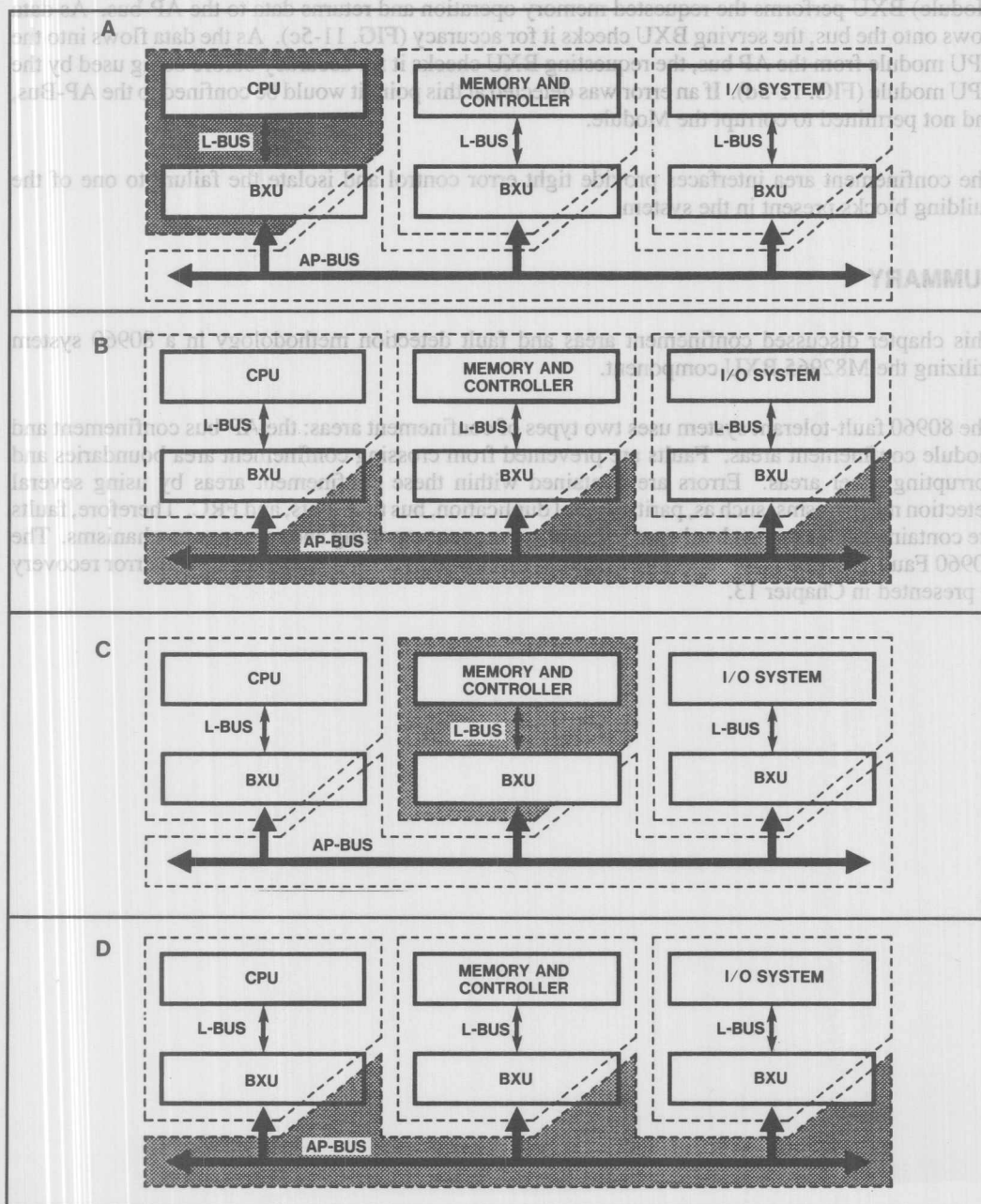


Figure 11-5: Confinement Area Operation

Before the information is accepted by the memory module, that module's BXU checks it for accuracy. If a failure is detected, it is confined to the AP-Bus because the information was valid when it left the CPU module confinement area (FIG. 11-5b). If no error is detected, the serving (Memory Module) BXU performs the requested memory operation and returns data to the AP-bus. As data flows onto the bus, the serving BXU checks it for accuracy (FIG. 11-5c). As the data flows into the CPU module from the AP bus, the requesting BXU checks it for accuracy before being used by the CPU module (FIG. 11-5d). If an error was detected at this point it would be confined to the AP-Bus, and not permitted to corrupt the Module.

The confinement area interfaces provide tight error control and isolate the failure to one of the building blocks present in the system.

SUMMARY

This chapter discussed confinement areas and fault detection methodology in a 80960 system utilizing the M82965 BXU component.

The 80960 fault-tolerant system uses two types of confinement areas: the AP-bus confinement and module confinement areas. Faults are prevented from crossing confinement area boundaries and corrupting other areas. Errors are contained within these confinement areas by using several detection mechanisms, such as, parity, signal duplication, bus time-outs, and FRC. Therefore, faults are contained at the lowest hardware level for subsequent reporting and recovery mechanisms. The 80960 Fault Tolerant error reporting system is discussed in detail in Chapter 12 and error recovery is presented in Chapter 13.

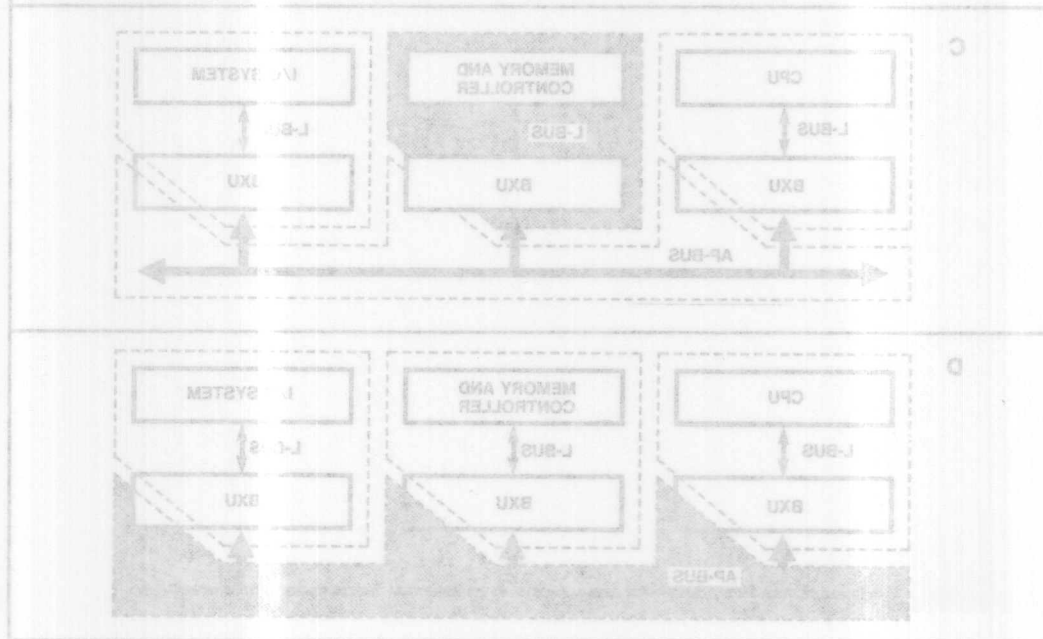


Figure 11-5: Confinement Area Operation

CHAPTER 12 ERROR REPORTING

Error reporting is the backbone of fault isolation and recovery. The error reporting system is designed so that, independent of the errors in the system, each BXU receives the same error report. When a BXU detects a fault, it broadcasts a serial error message to all other BXUs in the system. This error report message identifies the fault confinement area and the type and location of the component reporting the error. Sending this error report accomplishes two objectives: it informs the rest of the system that an error has occurred to prevent other confinement areas from using the inconsistent data, and it provides the necessary information for system recovery. Each BXU records this error message in the *Error-Log* register and proceeds independently with the appropriate recovery procedure.

This chapter describes the details of the error reporting procedure, specifically covering the following topics:

- The topology of the error reporting network
- The error reporting protocol including a description of how error reports are propagated throughout the system
- The error message format
- The types of error reports that are broadcast
- Error recording
- Error reporting diagnostics

TOPOLOGY OF THE ERROR REPORTING NETWORK

The serial error reporting network follows the same matrix topology as the AP-bus and L-bus. Two AP-bus error reporting lines ($\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$) and two L-bus error reporting lines ($\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$) are used to report the serial error messages. The $\overline{\text{LERL}}$ lines are connected to the other BXUs on the L-bus. Similarly, the $\overline{\text{BERL}}$ lines are connected to other BXUs on the AP-bus. An identical serial error report is sent over each of the pair of lines associated with a bus. One of the $\overline{\text{LERL}}$ lines may be connected to one of the active low interrupt pins of the 80960MC processor, so that the processor is notified of an error when it occurs.

The two error reporting lines that accompany each bus ensure that the error reporting network functions correctly independent of a failure. If either line should fail, the second line will continue to provide a correct error report. This level of redundancy ensures that the error report message reaches all components of the bus to prevent corruption of the other confinement areas.

Because the error reporting network is associated with multiple AP-buses (system can be configured with one, two, or four AP-Buses), another level of redundancy is reached. Failures in the error reporting network are either inside the AP-bus confinement area or the module confinement area. Thus, the replication of AP-buses and BXUs (a totally independent BERL network is implemented on each AP-Bus) provides another duplication of the reporting lines. As long as the system is

connected by functioning AP-buses, it also has a fully connected and functioning error reporting network.

ERROR REPORTING PROTOCOL

The error reporting protocol ensures that all components receive the error message in a timely manner. The reporting period lasts 256 AP-bus clock cycles and consists of two phases, as shown in Figure 12-1. The BXUs send the same error message twice during each phase. The error message consists of duplicate error reports.

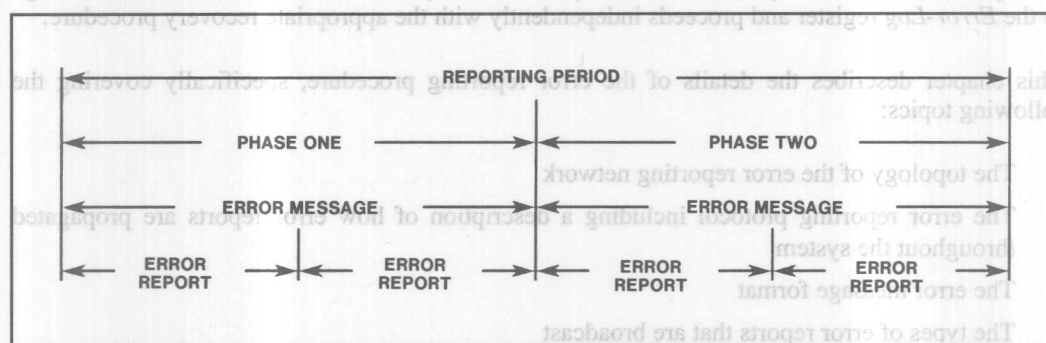


Figure 12-1: Error Reporting Period

If multiple error messages are broadcast, the protocol ensures that only one message is received by the components in the system. The description of the system-wide error reporting protocol is divided into the following three segments:

- $BERL_1$ - $BERL_0$ timing
- Phase one of error reporting
- Phase two of error reporting

$BERL_1$ - $BERL_0$ Timing

The BXUs send serial error messages at half the speed of the AP and L buses. Thus one bit is transmitted every two bus cycles (four CLK2 cycles). The error message begins with the *Start* bit followed by an additional 49 bits. This section provides a detailed description of the timing for the *Start* bit and the BXUs response to the start bit. The “Error Message” section of this chapter describes the contents of the error message.

The start of an error message indicates that all of the currently outstanding bus transactions should be retried because they may contain incorrect information. Because the AP-bus cycle is operating twice as fast as the error reporting cycle, the *Start* bit, which can occur on any AP-bus cycle, may

last for only half of the normal $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ bit time. The *Start* bit is only sent once; it is not repeated when the error message is sent again.

Two modes are available for controlling the timing of the BXU's reaction to errors, as shown in Figure 12-2. If the *BERL-Wait* bit in the *FTI* register (see Appendix A for the description of the *FTI* register) is set (called *Wait-For-BERL* timing), then the data received from the AP-Bus is held for 1.5 bus cycles by the BXU before the it places data on the L-bus. If the bit is not set (called *No-Wait-For-BERL* timing), then the incoming data will be held for only 0.5 bus cycles by the BXU before it places it on the L-bus.

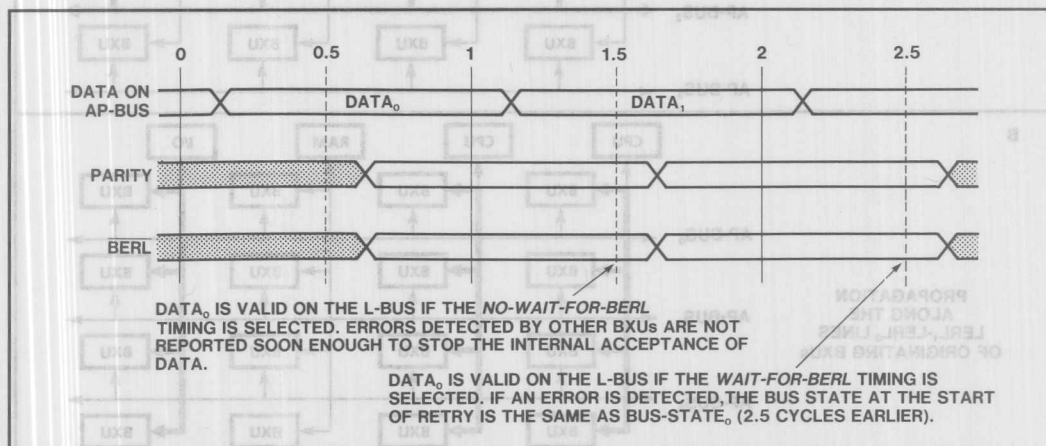


Figure 12-2: Timing for the Start of an Error Report

If the *No-Wait-For-BERL* timing mode is selected, the data is valid provided the following conditions exist:

- The $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines are not asserted 0.5 cycles after data.
- No parity error is detected.

If either a parity error is detected or either of the $\overline{\text{BERL}}$ lines were asserted, then this cycle is discarded, and the request is retried. This mode of operation allows correct recovery from parity errors detected by the BXU. FRC errors and parity errors detected by other BXUs are not reported soon enough to stop the internal acceptance of data. This provides higher performance in normal operation, but sacrifices some of the recovery capabilities that are available with the longer holdtime.

If the *Wait-for-BERL* timing mode is selected, then the $\overline{\text{BERL}}$ lines are checked 1.5 bus cycles after receiving data/address. If either of the $\overline{\text{BERL}}$ lines were asserted, then this cycle is discarded (data not placed on the L-Bus), and the request is retried. The reaction is always the same, because the extra cycle allows all decisions to be based solely on the state of the $\overline{\text{BERL}}$ lines, rather than using the internal parity information. This mode allows correct recovery from any type of error detected in the system.

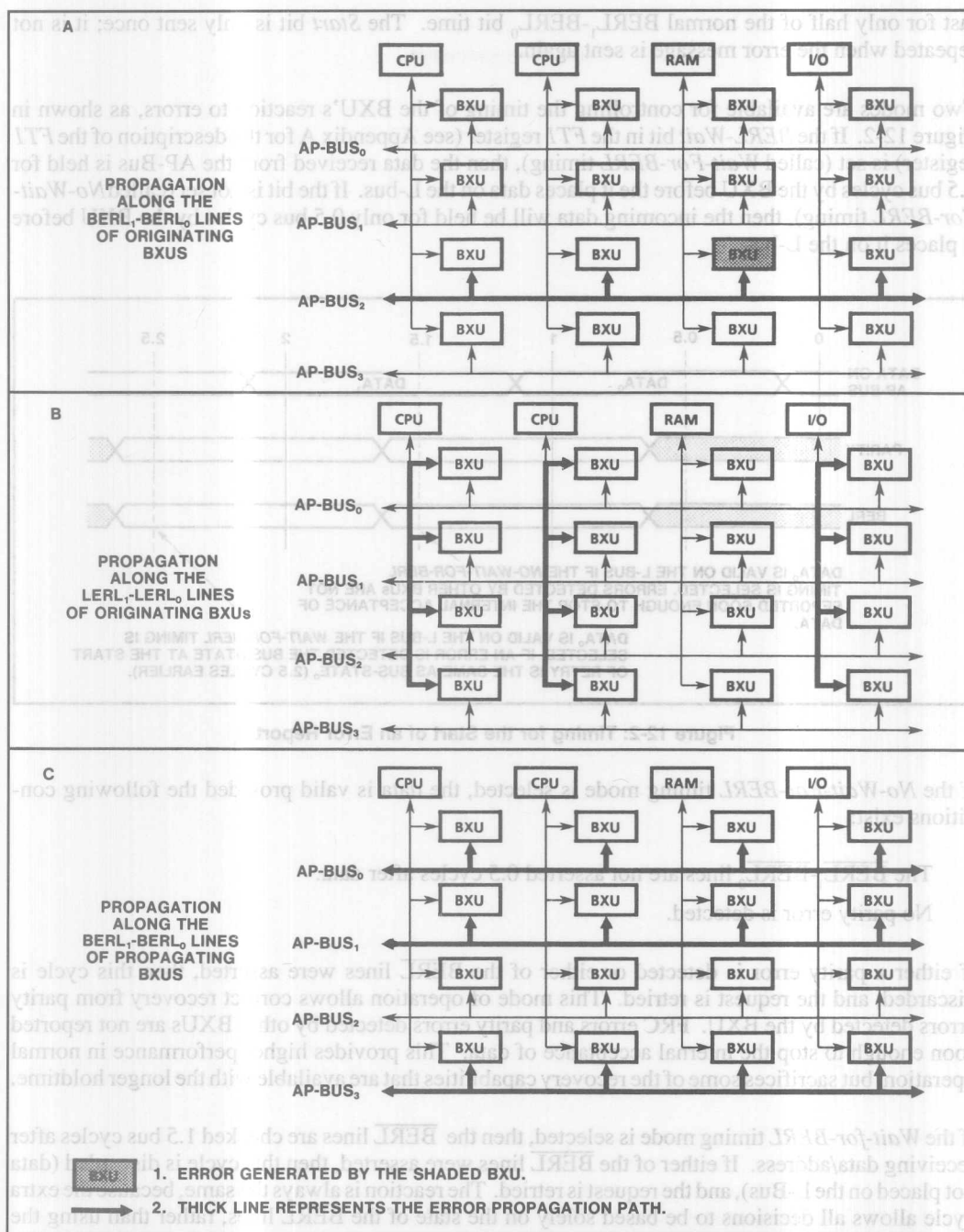


Figure 12-3: Error Report Propagation for Phase One

Phase One of the Error Reporting Sequence

During phase one of the error reporting cycle, the BXUs communicate the error message to each other throughout the system. The BXU detecting the error sends a serial error message along both $\overline{\text{BERL}}$ lines (see Figure 12-3a). This propagates the error message to all BXU's residing on the same AP-Bus as the BXU that detected the error. In the case of multiple errors being detected, the BXU generates only the highest priority error report on the bus. Those BXUs that detect an error message on their $\overline{\text{BERL}}$ lines first are called **originating** BXUs.

Four AP-bus clock cycles after the serial message started on the $\overline{\text{BERL}}$ lines, the originating BXUs in Processor mode issue an error report on both $\overline{\text{LERL}}$ lines on their L-buses (See Figure 12.3b). The BXUs that receive an error message on their $\overline{\text{LERL}}$ lines first are called **propagating** BXUs. The propagating BXUs send the error message on their $\overline{\text{BERL}}$ lines after a delay of four AP-bus clock cycles (See Figure 12-3c). All BXUs terminate their AP-bus operations upon receiving the *Start* bit of an error report message.

Figure 12-4 shows the timing of phase and its relationship to phase two (phase two is described in the next section). The propagating BXUs issue the error message eight AP-bus cycles after the originating BXUs during phase one. Whereas, the originating and propagating BXUs issue the error report at the same time during phase two.

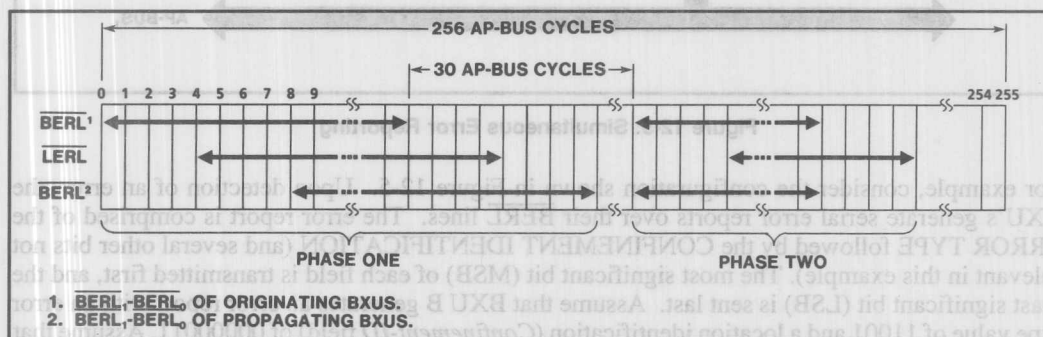


Figure 12-4: The Timing of the Error Report Propagation

The above scenario assumes that the BXU receives an error report from an active AP-bus, i.e., the *Inactive* bit of the *FTI* register was set to a value of zero. If the BXU receives an error report from an inactive bus, it does not propagate the error report from the $\overline{\text{BERL}}$ lines to the $\overline{\text{LERL}}$ lines (it ignores the error report). In the event that the BXUs must attach an inactive AP-bus, the BXUs transmit IAC requests from the $\overline{\text{LERL}}$ lines to the $\overline{\text{BERL}}$ lines to clear the *Inactive* bit in the *FTI* register.

If multiple error reports occur on the AP-bus, the BXU selects the first error report that it receives during an error reporting cycle. If a BXU receives an error report on its $\overline{\text{BERL}}$ lines, and simultaneously receives another error report on its $\overline{\text{LERL}}$ lines, it does not propagate either error report until phase two of the reporting sequence. (See the next section for the description of phase two.)

Several BXUs can react to a single error condition on either the $\overline{\text{BERL}}$ or the $\overline{\text{LERL}}$ lines and can simultaneously generate multiple error reports. In this case, the BXU propagates only the highest priority error report, which is based upon the contents of the error report message (see the “Error Message” section of this chapter).

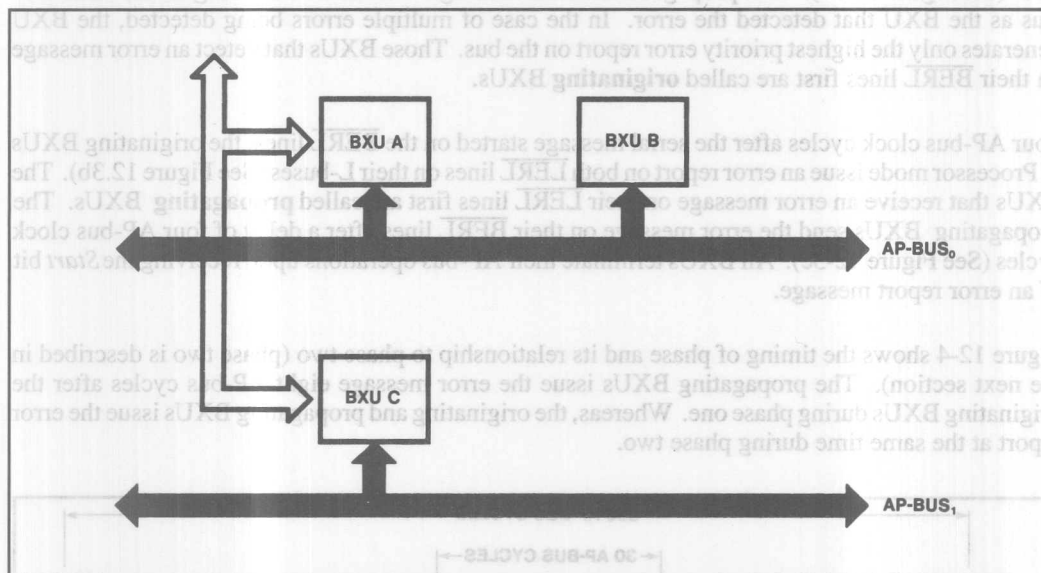


Figure 12-5: Simultaneous Error Reporting

For example, consider the configuration shown in Figure 12-5. Upon detection of an error, the BXU's generate serial error reports over their $\overline{\text{BERL}}$ lines. The error report is comprised of the ERROR TYPE followed by the CONFINEMENT IDENTIFICATION (and several other bits not relevant in this example). The most significant bit (MSB) of each field is transmitted first, and the least significant bit (LSB) is sent last. Assume that BXU B generates an error report with an error type value of 11001 and a location identification (*Confinement-ID* field) of 00000011. Assume that BXU C simultaneously generates an the same error report (an error type value of 11001), and it's *Confinement-ID* of 00000010. BXU B and BXU C simultaneously issue the error message on their respective $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines. BXU A receives the error message from BXU B and begins to propagate this error message on its $\overline{\text{LERL}}$ lines at the same time BXU C propagates its error message on the same $\overline{\text{LERL}}$ lines.

Each BXU continues to propagate the error message until a difference is detected. At this point, the BXU with the higher priority (the BXU with its $\overline{\text{BERL}}$ bit asserted) continues to propagate the message, while the other BXU begins to receive the message. In this example, both BXU A and BXU C propagate the error type and the first seven bits of the *Confinement-ID* because these values are identical. However, during the transmission of the eighth bit (the *Confinement* field LSB), BXU C continues to propagate its message, while BXU A begins to receive the error message.

At the end of phase one, BXU A and BXU C store the same error message in their *Error-Log* register, while BXU B stores the lower priority error message in its *Error-Log* register (see Appendix A for the description of the *Error-Log* register). This conflict is resolved in the same manner during phase two when the error message is broadcast again. This mechanism eliminates the lower priority error reports and makes sure that all components in the system receive the same error report message.

If a race (simultaneous reception) condition exists between error reports on the $\overline{\text{BERL}}$ lines and the $\overline{\text{LERL}}$ lines, the BXU ignores the message on the $\overline{\text{BERL}}$ lines and uses the information on the $\overline{\text{LERL}}$ lines, as this error report must have occurred prior to the error report just received on the $\overline{\text{BERL}}$ lines (the BXU does not propagate the error message received on the $\overline{\text{BERL}}$ lines).

For example, consider the configuration in Figure 12-5 again. For this case, assume that BXU C reports an error that occurred on AP-bus₁. BXU C issues an error message first on the $\overline{\text{BERL}}$ lines, then on the $\overline{\text{LERL}}$ lines four AP-bus clock cycles later. BXU A on AP-bus₀, in turn, propagates the error report eight AP-bus clock cycles later on their $\overline{\text{BERL}}$ lines.

Assume that BXU B reports a higher priority error that has occurred on AP-bus₀ simultaneously with the error message propagated by BXU A. Consequently, two error reports exist at the same time on the $\overline{\text{BERL}}$ lines associated with AP-bus₀.

During phase one, BXU B stores its error report in its *Error-Log* register, while the other BXUs store the error report propagated from the $\overline{\text{LERL}}$ lines in their *Error-Log* registers. During phase two, the first error report is issued again by all the BXUs because of the arbitration scheme. In this example, the error report from BXU C is acted upon and stored in all error log registers.

Phase Two of the Error Reporting Sequence

Phase two resolves any ambiguity that exists if multiple error messages were issued during phase one. If a BXU received at least one valid report message and the BXU is in Processor mode, then the highest priority message or first error message received during phase one is repeated. As shown in Figure 12-6, the error message is first repeated on the $\overline{\text{BERL}}$ lines, then the message is sent on the $\overline{\text{LERL}}$ lines. If a BXU did not receive any valid error messages, then it does not broadcast any error messages during phase two.

The BXUs in Memory mode do not send error reports during phase two because they are not required to be in a module that is fully connected to all AP-buses. Thus, these BXUs may have an incorrect view of the resolution of any conflicts between multiple error reports.

All BXUs in a subsystem receiving a valid error report issue their phase two report at the same time. Phase two begins 30 cycles after the end of the original report.

At the end of the error reporting cycle, the BXU copies the previous contents of the *Error-Log* register into the internal *Error-Record* register and places the last error message received in the *Error-Log* register (see Appendix A for the description of the *Error-Record* register). If the BXU does not receive any valid error messages, then the *Error-Log* register contains no useful information, and the

Invalid bit in the *Error-Log* register is set (this is not to be confused with the *Invalid* bit in the *Error Report*). The BXU operates its recovery algorithm on the final error message it receives, whether it was generated on its AP-bus or propagated from another bus.

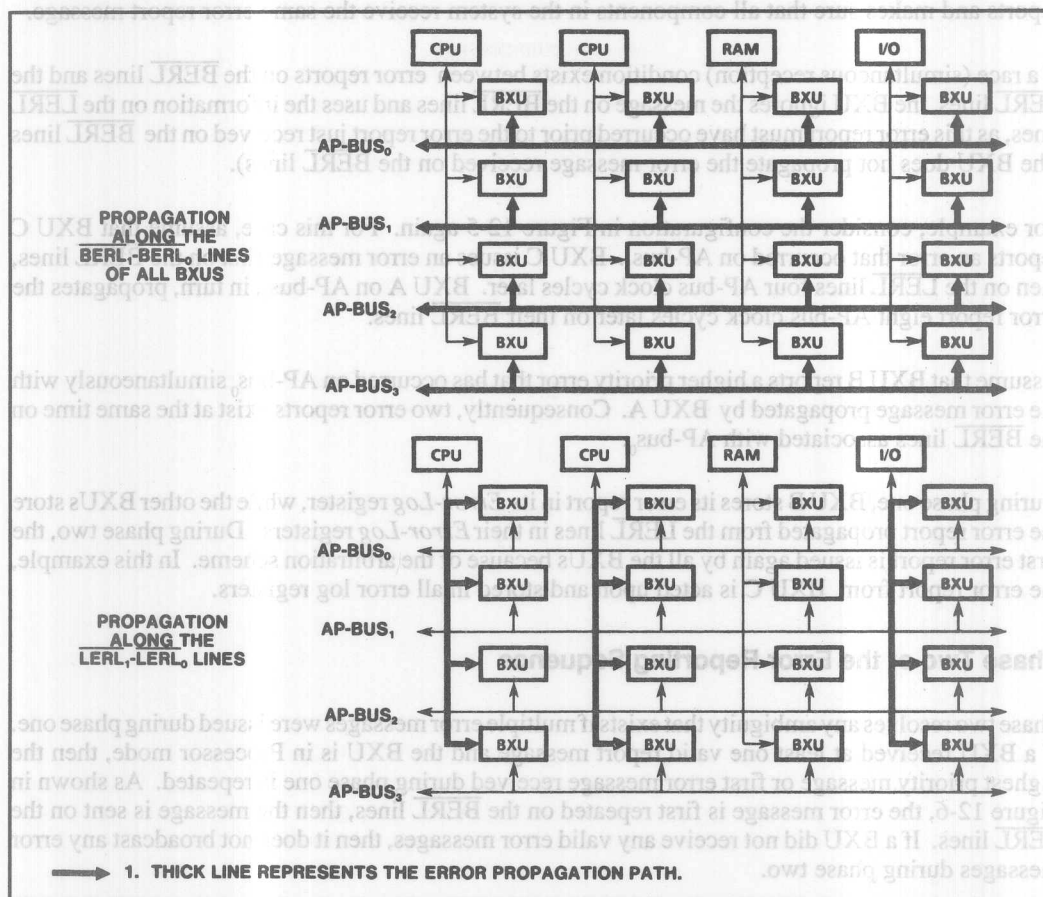


Figure 12-6: Error Report Propagation for Phase Two

ERROR MESSAGE FORMAT

Figure 12-7 illustrates the format and sequence for the 50-bit serial error message, which consists of a *Start* bit, *Null* bits, a *Error Type* field, a *Location-ID* field (comprised of the *Confinement-ID* and *Source-ID* sub-fields), an *Invalid* bit, and parity bits. The sequence of events for the error message is listed below:

1. Send a *Start* bit followed by two *Null* bits.

2. Broadcast the 22-bit error report twice to protect against burst errors.
3. After two *Null* bits, an *Invalid* bit if the error report has an error that did not occur in this confinement area.

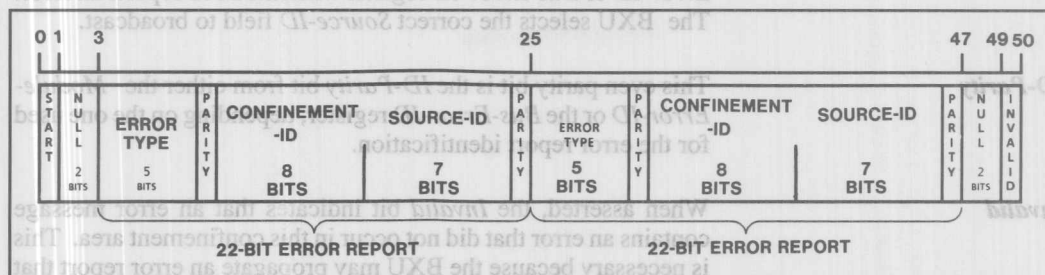


Figure 12-7: Error Message Sequencing

A description of each bit or field is provided in the following list. Note that the *Location-ID* field is further divided into the *Confinement-ID* and the *Source-ID* fields.

Start

This bit is used for two purposes: to indicate the start of an error message and to stop data processing. The *Start* bit (asserted low) marks the beginning of the reporting phase of a fault handling cycle.

Null

These two bits have a "don't care" value. This time period is used by the fault-tolerant logic of the BXU to process the error information and set up the propagation paths for the error report.

Error-Type

This five-bit field specifies the type of error that was detected. The field is fully encoded, providing up to 32 prioritized error types (only 15 are currently used): error type 31 is the highest priority and error type 0 is the lowest. The different error types are described in the "Types of Error Reports" section of this chapter.

Type-Parity

This parity bit provides odd parity over the *Error-Type* field. It does not cover any other bits in the message. This parity bit is generated internally by the BXU generating the error message.

Confinement-ID

This eight-bit field uniquely identifies a single confinement area where the error has occurred. The value for this field is the same as *Confinement-ID* field from either the *Module-Error-ID* register (for the module confinement area) or the *Bus-Error-ID* register (for the AP-bus confinement area) (see Appendix A for the description of the *Module-Error-ID* and *Bus-Error-ID* registers). The BXU selects the correct *Confinement-ID* field to broadcast.

Source-ID

This seven-bit field identifies the component reporting the error. This field is used by the diagnostic software to help isolate bus problems or provide recovery in self-healing systems. The value for this field is the same as the *Source-ID* field of either the *Module-Error-ID* or *Bus-Error-ID* register when the BXU reports an error. The BXU selects the correct *Source-ID* field to broadcast.

ID-Parity

This even parity bit is the *ID-Parity* bit from either the *Module-Error-ID* or the *Bus-Error-ID* register, depending on the one used for the error report identification.

Invalid

When asserted, the *Invalid* bit indicates that an error message contains an error that did not occur in this confinement area. This is necessary because the BXU may propagate an error report that contains errors.

TYPES OF ERROR REPORTS

The five-bit *Error-Type* field in the error report specifies what type of error has occurred. This section describes the details of the 15 types of errors that can occur.

Error Priorities

The following list consists of the errors reported by the BXU in priority order, the highest priority error is listed first. The number in parentheses is the priority number (in decimal notation). Each error type (noted by the sans serif *italics* typeface) identifies a faulty confinement area, as denoted by the *Location-ID* field in the *Error-Type* field description.

***Unsafe-Confinement-Area* (31)**

The BXU issues this report when a confinement area has an error that makes a retry operation dangerous. This is the highest priority error report and immediately causes a permanent error. The *Unsafe-Confinement-Area* error type is issued if the current error report has an *Error-Reporting-Error* associated with it and the previous report contains an *Error-Reporting-Error* type. The BXU determines the *Location-ID* field from either the *Bus-error-ID* or the *Module-Error-ID* register. If an error was detected on one of the BERL lines, the BXU uses the contents of the *Bus-Error-ID* register; if the error was detected on one of the LERL lines, the BXU uses the contents of the *Module-Error-ID* register. If the BXU that originates the error report detects an error on both error reporting lines, an *Unsafe-Confinement-Area* error type is issued using the contents of the *Module-Error-ID* register.

A BXU in Memory mode generates the *Unsafe-Confinement-Area* error type when the External Error (*ERR*) input signal is asserted. This type of error report may be used by external circuits to notify the system of an external fault. The contents of the *Location-ID* field are from the *Module-Error-ID* register.

A BXU generates the *Unsafe-Confinement-Area* error type when an FRC error is detected on the $\overline{\text{MODCHK}}$ pin. This is an unsafe error because the internal signal to initiate arbitration does not occur during the retry sequence. Hence, the module failure would be incorrectly reported as a bus failure. The contents of the *Location-ID* field are from the *Module-Error-ID* register.

Primary-Catastrophe (30)

The BXU generates this error report when it receives a *Primary-Catastrophe* command (see Appendix A for the description of the *Primary-Catastrophe* command). Software should issue this command whenever a condition external to the BXU indicates all primary units are about to fail, such as failure of the power supply for the primary units. This error report immediately causes a permanent error. The contents of the *Module-Error-ID* register are used for the *Location-ID* field.

Shadow-Catastrophe (29)

This error report is similar to the *Primary-Catastrophe* error report, except that it indicates that all shadow units are about to fail. The BXU generates this error report when it receives a *Shadow-Catastrophe* command (see Appendix A for the description of the *Shadow-Catastrophe* command). The contents of the *Module-Error-ID* register are used for the *Location-ID* field. This type of error report is treated in the same manner by hardware and software as the *Primary-Catastrophe* error report except that it pertains to shadow modules.

Error-Reporting-Error (28)

This report indicates that a component has detected a failure in the error reporting lines. The duplicate error reporting lines on each AP-bus allow the BXUs to issue this type of error report. The contents of the *Bus-Error-ID* register are used for the *Location-ID* field if the error occurs on the AP-bus, or the contents of the *Module-Error-ID* register if it occurs on the L-bus. The failure is caused by the following conditions: bad parity on either the $\overline{\text{BERL}}_1$ or the $\overline{\text{BERL}}_0$ line (or the $\overline{\text{LERL}}_1$ or the $\overline{\text{LERL}}_0$ line) in either copy of the error report, or a mismatch between the two copies of the error message within a phase of error reporting.

Bus-Arbitration (27)

This error report is issued when the BXU detects a bus arbitration error (by using FRC on $\overline{\text{BOUT}}$). The contents of the *Bus-Error-ID* register are used for the *Location-ID* field.

Bus-Parity (26)

This error report is issued when the BXU detects a bus parity error on the $\overline{\text{AD}}_{31}$ - $\overline{\text{AD}}_0$ or the $\overline{\text{SPEC}}_5$ - $\overline{\text{SPEC}}_0$ lines. The contents of the *Bus-Error-ID* register are used for the *Location-ID* field.

Component (25)

This error report is issued when the BXU detects an FRC error on the $\overline{AD}_{31}-\overline{AD}_0$ or $\overline{SPEC}_3-\overline{SPEC}_0$ pins on the AP-bus interface. The contents of the *Module-Error-ID* register are used for the *Location-ID* field.

Uncorrectable-Array-Error (23)

This error report is issued when an uncorrectable error is detected in the memory array. The BXU is informed of this condition when the external memory controller asserts the \overline{ECC} and \overline{UNC} pins. The contents of the *Module-Error-ID* register are used for the *Location-ID* field.

Correctable-ECC (22)

This error report is issued when a correctable error is detected in the memory array. The BXU is informed of this condition when the external memory controller asserts the \overline{ECC} pin, but not the \overline{UNC} pin, and the BXU is not asserting the \overline{COR} pin. ($\overline{ECC} \cdot \text{not } \overline{UNC} \cdot \text{not } \overline{COR}$) The contents of the *Module-Error-ID* Register are used for the *Location-ID* field.

COM-Altered (21)

This error report is issued by the BXU when the \overline{COM} input is toggled from high-to-low, and the *COM-Reporting* bit is set in the *FTI* register. This error report can be used by external hardware to notify the system of a change in status of the board. The contents of the *Module-Error-ID* register are used for *Location-ID* field.

Attach-Bus (19)

This error report is generated by the BXU in response to an *Attach-Bus* command. The contents of the *Bus-Error-ID* register are used for the *Location-ID* field (see Appendix A for the description of the *Attach-Bus* command).

Detach-Bus (18)

This error report is issued as a result of an *Detach-Bus* command. The contents of the *Bus-Error-ID* register are used for *Location-ID* field (see Appendix A for the description of the *Detach-Bus* command).

Terminate-Permanent-Error-Window (17)

This error report is issued as a result of a *Terminate-Permanent-Error-Window* command (see Appendix A for the description of the *Terminate-Permanent-Error-Window* command). The

contents of the *Module-Error-ID* register are used for the *Location-ID* field. Receiving this report ends the permanent error window period.

Sync-Refresh (16)

This error report is issued whenever the *Sync-Refresh* command is received by the BXU (see Appendix A for the description of the *Sync-Refresh* command). The contents of the *Module-Error-ID* register are used for the *Location-ID* field.

All error types use the *Module-Error-ID* register for the *Location-ID* field if the *Bus-Switch-Disable* bit in the *AP-Control* register is set at a value of one (see Appendix A for the description of the *AP-Control* register). This process provides the best chance for recovery in a single bus system. By taking the module out of operation, there is some probability that the rest of the system is able to continue. If the AP-bus were removed, then the system would stop operating.

No-Op (0)

Error type zero is reserved as a no-op because zero is the default value loaded into the *Error-Log* Register at RESET.

Error Types for Different Detection Mechanisms

Table 12-1 lists the *Error-Type* field that is generated by each of the detection mechanisms in the BXU.

It is possible for a single failure to be detected as multiple error types by different units on the bus. Error types are prioritized to make sure that the location of the failure is correctly identified. When multiple errors are detected (or reported), the error report with the highest priority error is selected by the BXU for recovery.

Using Commands to Generate Error Reports

Some of the error types are generated as a result of commands to the BXU. Prior to issuing the command to the BXU, the *Testing-Enable* bit in the *Test-Detection* register must be set (see Appendix A for the description of the *Test-Detection* register). Before setting the *Testing-Enable* bit, there should be a single bit with a value of one in the *COM* register. When the command is executed, the *Testing-Enable* bit is cleared. If latency is important, then the *Testing-Enable* bit may remain set. In this way, no extra accesses are required at the time of the command.

Table 12-1: Error Types for Detection Mechanisms

Detection Mechanism	Error Type
FRC on AD or SPEC signals	Component
FRC on $\overline{\text{BOUT}}$ signal	Bus-arbitration
FRC on $\overline{\text{MODCHK}}$ signal	Unsafe-confinement-area
AP-bus parity	Bus-parity
Error on $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ or $\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$	Error-reporting-error
Error on $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ or $\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$ with error-reporting-error in error-log register having the confinement-ID field of this BXU's AP-bus or module, respectively.	Unsafe-confinement-area
$\overline{\text{COM}}$ input pin	Com-altered
$\overline{\text{ECC}}$ pin ($\text{ECC} \cdot \overline{\text{UNC}} \cdot \overline{\text{COR}}$)	Correctable-ECC
$\overline{\text{ECC}}$ and $\overline{\text{UNC}}$ pin	Uncorrectable-array-error
$\overline{\text{ERR}}$ pin	Unsafe-confinement-area

ERROR REPORT LOG

Two registers are used in recording the error report: the *Error-Log* register contains the current error report and the *Error-Record* register contains the previous error report. When an error report is received, the contents of the *Error-Log* register are copied into the *Error-Record* register. The *Error-Log* register is accessible to software and represents the primary form of communication between the hardware fault-handling mechanisms and the software routines responsible for system management. The information contained in the *Error-Log* register always reflects the latest error report generated in the entire system.

The *Error-Log* register is used independently by the hardware and software. The hardware recovery procedures use the *Error-Type* and *Location-ID* fields in determining which recovery actions are performed. The hardware reacts immediately to each error report; thus, it never encounters an overflow condition.

To determine if the current error is permanent, the BXU utilizes the *Error-Record* register. The error is treated as permanent if the contents of the *Error-Record* register match the *Error-Log* register after a retry operation.

The BXU provides a software-visible, as well as an external indication of a permanent error in a module. Software can read the *Error-Log* registers to determine whether an error is permanent. The

BXU asserts the COM pin to indicate a permanent error in the hardware system. For details on COM pin activation, see the “Serial COM Protocol” section in Chapter 14.

ERROR REPORTING DIAGNOSTICS

The internal fault-tolerant logic of the BXU provides all the standard error checking and reporting capabilities that are communicated over the $\overline{\text{BERL}}$ lines. This logic and the error reporting network, however, are also susceptible to failures. The BXU takes care of this situation by making provision for diagnostic tests of the fault-tolerant logic and the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ network. This section examines these cases.

$\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ Error Detection

Because the error reporting system is fundamental to the correct operation of a fault-tolerant system, error detection mechanisms are built into each error report network. These mechanisms ensure that the error report network operates correctly in the presence of single failures. The $\overline{\text{BERL}}$ lines use the following two error detection mechanisms:

1. Parity on the error message (*Type* and *Location-ID* fields)
2. Error report duplication (The sequence is repeated twice for a single error report).

These mechanisms detect an error in the error reporting network, itself. The Invalid bit at the end of the error report is used by the BXU to indicate whether the error reporting error originated in this confinement area or not. The BXU that originates the error report always clears the *Invalid* bit. If the BXU that propagates the error report detects an error in the error message, it performs one of the following actions.

1. If the *Invalid* bit is cleared, the BXU knows that the error originated in the confinement area associated with the error report lines that the error report was received on. It sets an internal flag and sets the *Invalid* bit in both of the error reports on the $\overline{\text{BERL}}$ and $\overline{\text{LERL}}$ lines that it is in the process of retransmitting (one *Invalid* bit for each error reporting line) that it is in the process of retransmitting.
2. If the *Invalid* bit of the incoming error report is set, it knows that error reporting error did not originate in one of this BXU's confinement areas. The BXU sets the *Invalid* bit in both of the error reports on the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ and $\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$ lines that it is in the process of retransmitting (one *Invalid* bit for each line).

In either case, the BXU participates in phase two of the error reporting if at least one report is error-free. An *Error-Reporting-Error* is marked as a permanent error, if it occurs within the permanent error window and in the same confinement area as the last *Error-Reporting-Error*. BXU's that detect error reporting errors do not participate in retry, but instead transmit the “Error Reporting Error” error report at the end of the transient waiting period.

Fault-Tolerant Logic

In addition to its error reporting network diagnostic capabilities, the BXU provides diagnostic support for the fault-tolerant logic. The following sections describe the accessibility of the fault-tolerant logic, the parity logic test, FRC logic test, bus time-out logic test, and the error reporting logic tests.

Control and Accessibility

The control of which BXU in a logical module responds to fault tolerance diagnostic testing is provided by the *Access* bits in the *FRC* and *QMR* registers (see Appendix A for the description of the *FRC* register and the *QMR* register). Software sets the *Access* bits to point to the selected BXU.

The accessibility to the fault-tolerant logic of the BXU is through the *Test-Detection* register and *Access* mode of logical IAC type 0010_B (Register Request Using a Logical Address). By writing a *Testing enable* bit to the *Test-Detection* register with a Logical IAC in the *Access* mode, only the BXU that the *FRC* and *QMR* registers *access* bits are pointing to is selected. This selection is important for parity testing. For instance, when parity testing is performed, the parity information must be corrupted for only one BXU in order to detect if the circuitry is properly functioning. By enabling the *Access* bit with this type of IAC, the software can specify any master or checker in the primary or shadow module to perform the test. In this case, the other BXUs in the primary and the shadow modules perform dummy operations to maintain lockstep operation.

The *Access* bit can only be set if the IAC request is a Write Request to a AP-Bus register in another module. If a 80960MC processor sends an IAC to a BXU in its own module, the *Access* bit is ignored. Also, the *Access* bit will only control which BXU replies on the bus if both the toggle bits are off in both the *FRC* and *QMR* registers.

Parity Logic

The parity checking circuitry for each parity pin can be individually or jointly exercised by using the *Parity-Test* field of the *Test-Detection* register. The *Parity-Test* bits selectively force a parity error by inverting the *CHK* pin(s) sampled from the bus. This action will create a parity error, which the parity comparing logic should detect. The BXU reports an error if the *Parity-Test* bit is set and a parity error is detected. Parity testing can be performed on-line, but a real parity error (i.e., non-forced) during the test will go undetected. The success of the test is noted by checking the *Error-Log* register after the test.

The *Testing-Enable* bit in the *Test-Detection* register must be set before executing parity testing. Before setting the *Testing-Enable* bit, there should be a single bit with a value of one in the *COM* register. This is a condition for any test that uses the *Test-Detection* register.

FRC Logic

The FRC circuits are self checking. To initiate FRC testing, software must perform the following actions:

2. Set the *Testing-Enable* bit in the *Test-Detection* register.

Once testing is started, the FRC circuits do not require any special test sequences to check their operation. The testing proceeds only as long as the *Testing-Enable* bit is set.

Bus Time-Out Logic

The bus time-out timer can be checked by performing an AP-Bus read operation to a memory location that has no BXU assigned to it. A read request is used so that the 80960MC processor receives a direct indication (BADAC asserted) that the time-out worked correctly.

Error Reporting Logic

The *Test-Report* command forces the BXU to test the error reporting network of the AP-Bus by generating an error report (see Appendix A for the description of the *Test-Report* command). The type of error report is specified by the bit pattern in the *Test-Type* register (see Appendix A for the description of the *Test-Type* register). Multiple errors may be loaded into the bit pattern to test the priority circuits in the fault-handling logic. After issuing the command, the software reads the *Error-Log* register to check that the error was properly handled.

The *Testing-Enable* bit in the *Test-Detection* register must be set before executing the *Test-Report* command. Before setting the *Testing-Enable* bit, there should be a single bit with a value of one in the *COM* register.

Handling Errors in the Fault-Tolerant Logic

Errors may occur in the fault-tolerant logic. By latent fault testing, the BXU can detect these errors, if any, and respond to them appropriately. The possible errors that may occur in the fault-tolerant logic are listed below.

Detection Mechanisms. If the FRC detection mechanism that is internal to the BXU fails while asserted, then the failure is handled correctly as an error. If this mechanism fails while not asserted (the internal drivers are open), then the failure is undetected. Latent fault testing can be performed by software to check for this failure.

Sensing and Encoding the Error Type. If this internal logic of the BXU fails to sense the error type properly or report the wrong error type while asserted, then the failure is handled correctly as an error. If the logic fails while not asserted (the internal drivers are open), then the failure is undetected. Latent fault testing is provided by using the *Test-Report* command and *Test-Type* register.

Module-Error-ID and Bus-Error-ID Registers. A software-loaded parity bit detects any odd number of bit errors in the identification information sent on the AP-bus (see mechanism 1, which

is described later in this section). Errors not detected by the parity bit could cause system failure. Latent testing can be performed using the *Test-Report* command and *Test-Type* register.

Start Bit. If the *Start* bit fails while asserted, the failure is handled correctly (see mechanism 1 below). If the *Start* bit fails while not asserted (the internal driver is open), then the failure is undetected. Latent fault testing can be performed by using the *Test-Report* command and *Test-Type* register.

Error Report Generation. Bits that remain asserted are detected by using parity and signal duplication. The recovery from this situation uses mechanism 1 listed below. An open line causes an *Error-Reporting-Error* error message.

Error Report Propagation. The error report flows through two totally independent paths in the BXU. Propagation errors are handled in the same manner as the errors on the $\overline{\text{BERL}}$ and $\overline{\text{LERL}}$ lines with internal fully redundant paths.

Recovery Actions. Errors in recovery are caught by FRC disagreement between the master and checker. Latent testing is possible by forcing recovery actions to take place.

Two special mechanisms are provided to detect and recover from some of the error cases described above.

Mechanism 1. This mechanism splits the $\overline{\text{BERL}}$ lines from the master and checker to prevent a failed BXU from corrupting both $\overline{\text{BERL}}_1$ and $\overline{\text{BERL}}_0$. The master only drives $\overline{\text{BERL}}_0$ and the checker only drives $\overline{\text{BERL}}_1$ if any of the following conditions exist:

- Neither $\overline{\text{BERL}}_1$ or $\overline{\text{BERL}}_0$ have a good report at the end of phase one or phase two.
- The *valid* bit is not set in the *Error-Log* register at the beginning of retry
- The *valid* bit is not set in the *Error-Log* register when software writes to it (testing purposes only)

This action allows the other BXUs to issue an *Error-Reporting-Error* and prevents the faulty BXU from corrupting both $\overline{\text{BERL}}$ lines. This mechanism can be tested by clearing the *Valid* bit and sending a test report. If the mechanism works, then an *error-reporting-error* should be recorded (since the message would only go out on one $\overline{\text{BERL}}$ line instead of both of them).

Mechanism 2. This mechanism prevents the system from suspending activity. An *Unsafe-Confinement-Area* type error report is issued, if the following two conditions exist:

- The current error report has an *Error-Reporting-Error* type associated with it.
- The previous report contains an *Error-Reporting-Error* type.

This mechanism avoids infinite loops alternating between a corrupt error reporting and the reporting of the corruption.

SUMMARY

Error reporting is the link between detection and recovery. The 80960 system uses a well defined protocol to report errors. This protocol defines the timing and error propagation methodology. The dual serial error reporting networks use the $\overline{\text{BERL}}$ and $\overline{\text{LERL}}$ lines to ensure that the error report is broadcast to every BXU in the system. The protocol takes into consideration the situation of multiple and simultaneous error reports. The error report format consists of a 50-bit message that incorporates redundancy and parity to ensure a correct report.

SUMMARY

Error reporting is the link between detection and recovery. The 80960 system uses a well defined protocol to report errors. This protocol defines the timing and error propagation methodology. The dual serial error reporting networks use the BERL and LERL lines to ensure that the error report is broadcast to every BXU in the system. The protocol takes into consideration the situation of multiple and simultaneous error reports. The error report format consists of a 50-bit message that incorporates redundancy and parity to ensure a correct report.

CHAPTER 13 RECOVERY

Previous chapters explained the 80960 architecture's mechanisms for detecting and isolating failures to a confinement area, and reporting those failures throughout the system. This chapter describes the architecture's hardware recovery mechanisms that are totally transparent to software. For example, when transient errors occur, the requests are automatically retried; when permanent errors occur, the hardware automatically removes the faulty module or bus, and reconfigures the system with redundant resources.

The recovery algorithm is executed in parallel by all BXUs in the system. No global agent is responsible for correct recovery actions. Each BXU performs its recovery sequence independently from all other BXUs in the system. This distributed approach to error reporting ensures that no single component failure, as would be the case with a global agent, can bring the total system down.

Figure 13-1 provides an overview of the error recovery procedure. Error detection and reporting, which were discussed in previous chapters, are included in the diagram for completeness. The recovery process encompasses the five major areas required for hardware fault recovery listed below:

- Types of Errors
- Unsafe Error Decision
- Retry Sequence
- Permanent Error Decision
- Resource Reconfiguration

TYPES OF ERRORS

The details of how these recovery actions operate are presented in separate sections. Before examining the recovery mechanisms, however, it is important to clearly define transient errors and permanent errors. Transient errors are often generated by electromagnetic noise bursts caused by some nearby device, such as a large motor starting or a relay de-energizing. Errors of this type require recovery (correction), but do not require a system hardware reconfiguration. However, permanent errors are indicative of a failed module or AP-Bus, and will require the hardware to isolate the defective confinement area and shift its activities to a back-up resource.

Transient Errors

An error is classified as a transient error, if **both** of the following conditions exist:

- The error type is not an *Unsafe-Confinement-Area*, *Primary-Catastrophe*, or *Shadow-Catastrophe*
- The error type or the confinement area in the current error report is different from the error type and the confinement area contained in the *Error-Record* register.

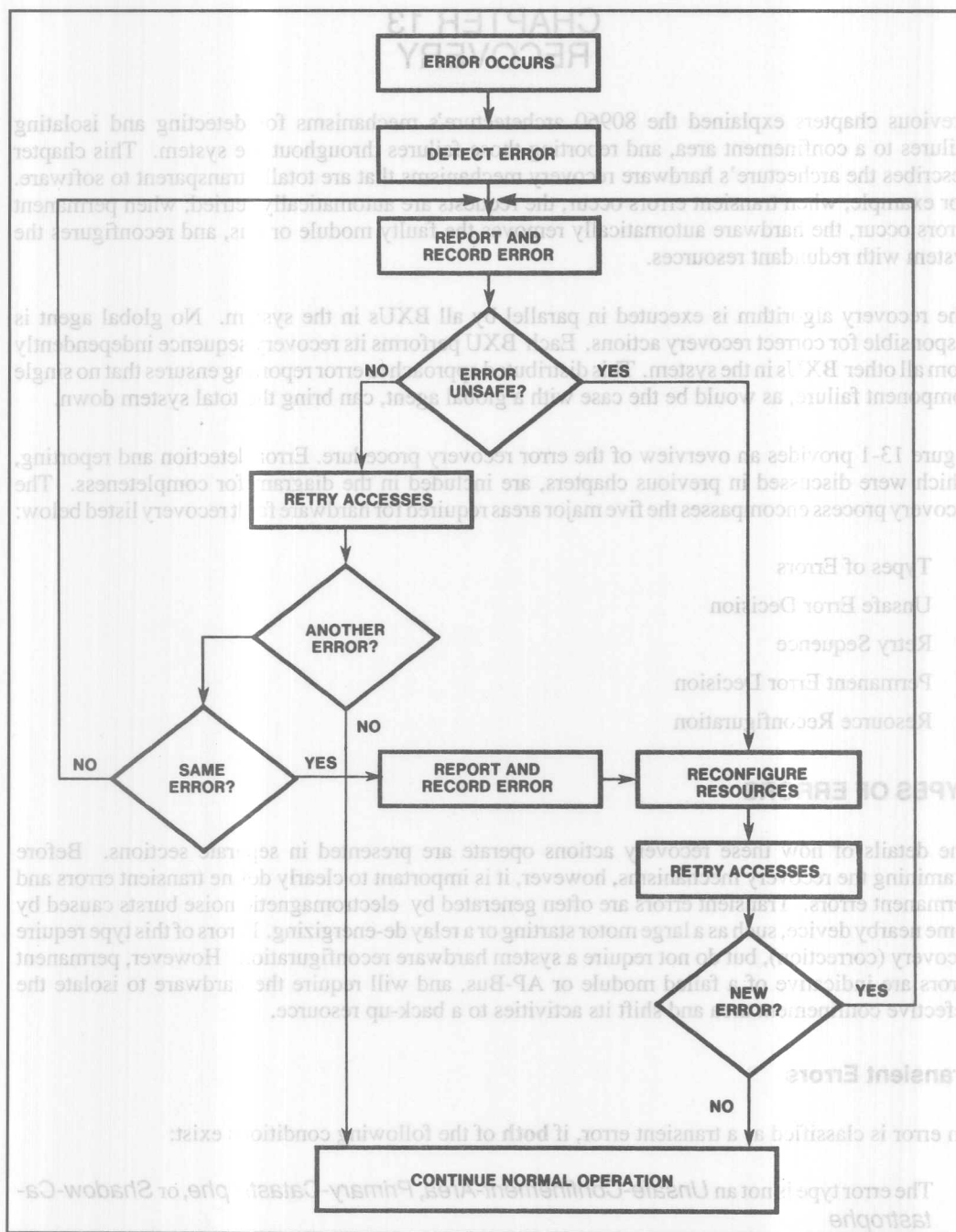


Figure 13-1: Recovery Procedure

Permanent Errors

An error is classified as a permanent error, if **either** of the following conditions exist:

- The error type is an *Unsafe-Confinement-Area*, *Primary-Catastrophe*, or *Shadow-Catastrophe*
- The error type and the confinement area in the current error report are the same as the error type and the confinement area contained in the *Error-Record* register.

When a permanent error is reported, the BXUs take special actions to recover from the failure. In all cases, the BXUs set the *Permanent-Error* bit in the *Error-Log* register. Additional recovery actions are taken by the BXU if the error occurred in its confinement area, or its backup/spouse's area.

UNSAFE ERROR DECISION

The recovery sequence is the same for most error types except for the following three: *Unsafe-Confinement-Area*, *Primary-Catastrophe*, and *Shadow-Catastrophe*. These types of errors are handled immediately as permanent errors to prevent recursive error reporting cycles. For all other types of errors, retry is the first step in the recovery sequence.

The BXU uses the *Error-Log* and *Error-Record* registers to make the unsafe error decision. The *Error-Log* register contains the current error report and the *Error-record* register contains the previous error report. If the error type is considered unsafe, either because it was one of the three immediately handled types described above or the two error registers match, then the BXU will directly enter the reconfiguration sequence. In all other cases, the BXU will begin the retry sequence.

RETRY SEQUENCE

The first step toward recovery is to retry the outstanding AP-Bus accesses. This section describes the retry sequence and explains some considerations as a result of the retry sequence.

Retry Operation

When a 80960MC processor makes a request, all information associated with the request (address, control, data) is automatically stored in the retry buffers of the BXUs. This information is held until the access is finished. All accesses that were outstanding at the time of an error will be retried. Storing the access request in retry buffers adds no delay to the access latency of a 80960MC processor request.

If a retry sequence is necessary, the BXU must arbitrate again for access to the AP-bus. The outstanding requests of the BXU are retried in the same order as they were issued on the L-bus. This is true even if bus switch occurs because partner BXUs keep track of the state of requests being handled by their partners (see the "Bus Switching" section for details on bus switching). However, because the BXUs arbitrate for the AP-bus again, these requests may not maintain the same order on

the AP-bus with respect to requests issued by other modules. While the BXU retries it requests, it does not operate on new requests (an address is accepted, but the data is not processed).

The retry buffers can be tested for correct operation by issuing a *Test-Report* command to any BXU in the system (see Appendix A for the description of the *Test-Report* command). If any buffer has failed, it will be detected as a FRC error in the module.

Special Considerations for the Retry Function

When the BXU retries requests, special considerations are made for three cases: completion of operations buffered by the BXU, multi-word outbound requests, and cache operations after a retry sequence.

Completion of Inbound Operations

Operations that are buffered by the BXU in the incoming direction are completed, regardless of retries on the AP-bus, if all data was received correctly into the buffer. These buffered operations include the following:

- RMW-Read requests
- Memory write operations if the BXU is acting as a memory controller

In other words, once the data for these buffered operations is correctly received from the AP-bus (i.e., without errors being reported on the AP-bus $BERL_1$ - $BERL_0$ lines), and the reply cycle is completed, the BXU finishes the operation on the L-bus. This process occurs no matter what happens on the AP-bus.

For example, the BXU in Memory mode ensures that all write requests are atomic (either the transactions are completed or they are not performed) by fully buffering the write request before initiating any action on the L-bus. Once the L-bus operation is started, the BXU ensures that it is able to write the correct values into the memory in this module.

Multi-Word Outbound Read Requests

Multi-word outbound read requests may return inconsistent data if these requests are accessing locations that are being shared without the protection of hardware or software locks.

The following example shows how a read request can return inconsistent data.

- Assume that the 80960MC processor performs a four-word read operation. The BXU accepts the read request and issues it on the AP-bus. The L-bus is held, by inserting wait states, until the data returns.
- The data starts to return from the AP-bus and the BXU passes it onto the L-bus to maintain performance. The 80960MC processor accepts two words of the four-word request.

- An AP-bus error forces a retry and the read request is reissued by the BXU.
- This time, however, the read request loses arbitration and a write request from another module writes to the same location that is accessed by the read request.
- The retried read request now accesses updated locations and the last two words of the request are transferred to the 80960MC processor on the L-bus. However, the first two words are inconsistent with the last two words of the read data.

In summary, single-word read operations are always indivisible with respect to write operations. However, multi-word read requests are not indivisible unless they are cached.

Cache Considerations with Retry

If an error is reported during a cache fill request, the cache write operation always is restarted from the beginning of the request after the retry sequence. Thus, if an error cancels an AP-bus reply after two of the four words were written into the cache, the two words will be rewritten when the request is retried. Starting the cache write operation from the beginning ensures that cache fill requests occur from a single AP-bus transaction. Consequently, the cache always holds consistent data.

PERMANENT ERROR DECISION

Permanent errors are defined in one of two ways: either the error type identifies the fault as a permanent error, or a transient error occurs twice in a row. Since the definition of a permanent error is receiving the same error report twice in a row, there needs to be a way to prevent the hardware from labeling two errors that were hours apart as a permanent error. This is done with the *Terminate-Permanate-Error-Window* Command. The permanent error window begins coincidentally with the retry sequence. The period ends when software issues a *Terminate-Permanent-Error-Window* command (see Appendix A for the description of the *Terminate-Permanent-Error-Window* command). The BXU responds to this command by sending an error report using the *Terminate-Permanent-Error-Window* error type. Thus, the *Terminate-Permanent-Error-Window* error type provides the capability to set the time period in which the error can occur again.

The *Terminate-Permanent-Error-Window* error type clears the *Error-Count* field in the *Error-Log* register in all BXUs. If *Terminate-Permanent-Error-Window* error report is received twice in a row, it is labeled a permanent error, and the normal permanent error operations occur (i.e., the module is removed from the system since the *Terminate-Permanent-Error-Window* error report uses the *Module-Error-ID* register as its location identification). Because the *Terminate-Permanent-Error-Window* error report results from a software command, it can be used as a delimiter between error reports resulting from hardware error detection mechanisms.

RESOURCE RECONFIGURATION

Resource reconfiguration provides a way to recover from permanent errors. Once a faulty confinement area is known, the hardware system can automatically replace this area with a properly

functioning one, assuming that there are redundant resources available. To implement this, the 80960 hardware system uses the following recovery mechanisms.

1. Module shadowing
2. Bus switching
3. FRC splitting

This section examines the details of each of these recovery mechanisms. Before explaining these items, however, the redundant resources are described to provide background information.

Redundancy

Recovery from permanent hardware failures can take place with redundant resources without any impact on logical software operation. With redundant resources, each of the self-checking modules and buses can be paired with an identical back-up module or bus. This feature, module shadowing, allows continued operation in the presence of any single failure. Modules can be paired with any other identical module. No predetermined electrical connections are required; shadowing is done by means of logical configuration information. The two modules in the pair then operate in lock step, providing a complete and current backup of all state information in the case of a failure.

Buses can also be paired together to provide redundant bus channels. Since buses do not hold state information, both buses in a pair may be used to carry traffic during normal operation. Thus, the back-up bus capability can be used to increase total bus bandwidth. The mechanisms for establishing and maintaining redundant operation reside totally in the BXU. The 80960MC processor is unaware that it is operating in a redundant configuration.

There is an important distinction between redundant resources and alternate resources. Redundant resources provide a complete and current backup for the resources in the system. When a failure occurs, the backup can mask the fault from the rest of the system. This recovery occurs automatically under the control of the hardware. Alternate resources provide multiple modules capable of performing the same tasks, but not a complete and current backup for the resources in the system. Alternate resources allow reconfiguration around failures, but do not mask the failure from the rest of the system. Alternate resources can be used to achieve deferred maintenance capabilities. An example of this type of configuration was given in Figure 10-7.

The mechanisms and resources used for redundancy are orthogonal to the mechanisms and resources used for detection. Moreover, adding redundancy for recovery does not degrade the detection capabilities or performance characteristics of the system.

Processor Module Shadowing

The concept of module shadowing requires redundant resources in the form of two identical modules. The two modules must be operated in lock-step. One module is the active unit, or primary unit, and the other is the passive unit, or shadow unit. The module shadowing configuration is illustrated in

Figure 13-3 shows an example of a married processor module configuration. In this example, all the components attached to the L-bus are placed in the same state (arbitrarily called state “ZZZ”). To ensure that the BXUs are in the same state, the registers of the QMR module are set to same value except those that are shown (the contents of each register are represented by the three capital letters). The *Bus-Error-ID*, *Module-Error-ID*, and *Physical-ID* registers of each FRC pair have different values to uniquely identify the FRC pair. The contents of the *Spouse-ID* register contain the contents of the respective *Module-Error-ID* register of the FRC pair. These settings of the *Spouse-ID* register make each FRC pair aware of the other pair. The processor modules are married by setting *Married* bit of the *QMR* register in both FRC pairs and the *Shadow* bit of the *QMR* register only one FRC pair. The two married FRC modules form a single QMR logical module.

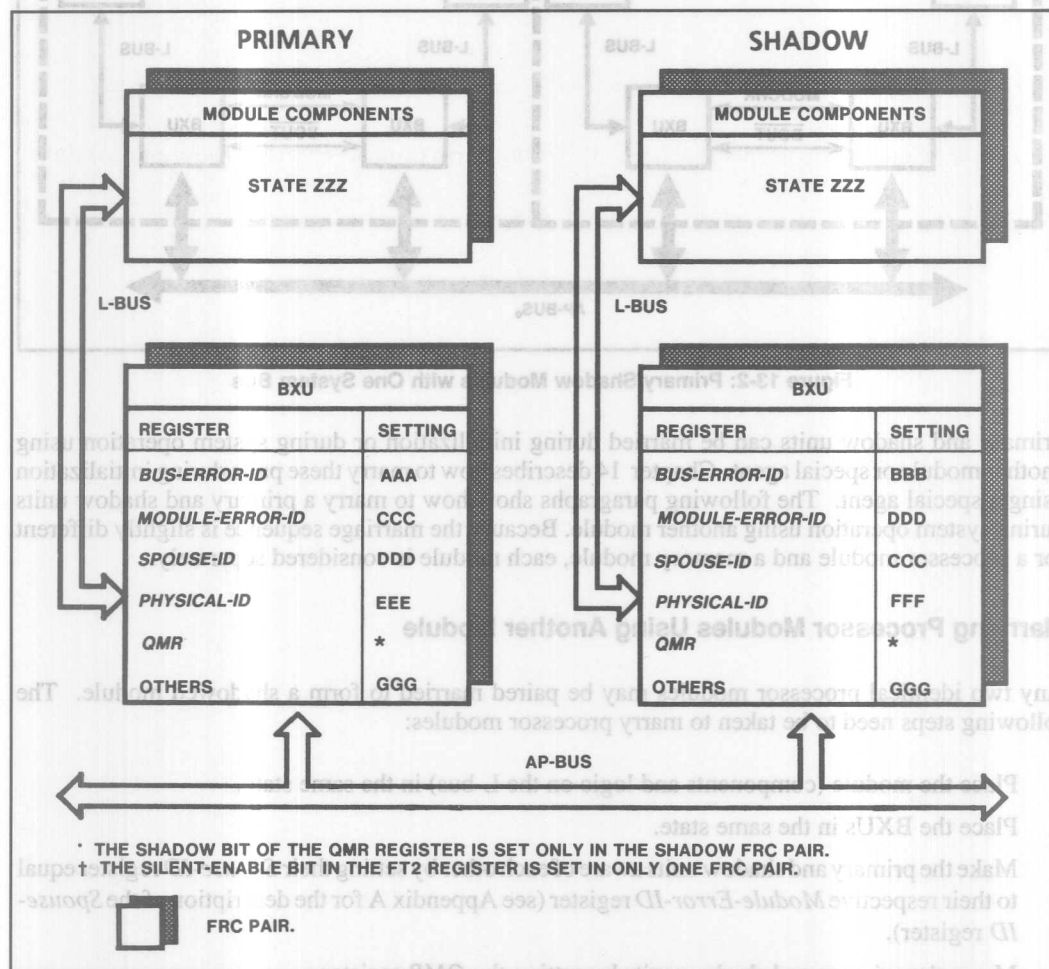


Figure 13-3: Example of Married Processor Modules

The following enumerated list is an example of one possible way (out of several) to marry the primary and shadow units. The module that becomes the primary unit is labeled the *Primary-Elect* unit, and the shadow unit, *Shadow-Elect* unit. If there are multiple buses in the system, any action performed on one BXU in a module is performed on all BXUs in that module.

1. Remove both processors from the pool of active processors available for work.
2. Stop references to/from the components attached to the L-bus of the BXU in the *Shadow-Elect* unit by performing the following actions:
 - a. Send the 80960MC processor(s) a "Stop-Processor" IAC message to stop the 80960MC processor(s) in the *Shadow-Elect* unit.
 - b. Set the *Shadow* bit in the QMR register of the BXU, effectively making this the *Shadow-Elect* unit.
 - c. Disable the AP-bus address recognizer of the BXU by clearing the *AP-Match* register.
3. Ensure that the components attached to L-bus of the BXU in the *Primary-Elect* and *Shadow-Elect* units are in the same state by performing the following actions:
 - a. Stop the 80960MC processor(s) in the *Primary-Elect* unit, effectively synchronizing the FRC pair.
 - b. Disable the AP-bus address recognizer of the BXU's by clearing the *AP-Match* register.
 - c. If there is local memory on the BXU's L-buses, it must contain the same data in the *Primary-Elect* and *Shadow-Elect* modules. This is done by copying the contents of one unit to the other. The *Primary-Elect* and *Shadow-Elect* units can ensure that memory has the same data by setting the AP-bus address recognizers to different values (both unused by the system currently in operation), then copying the contents of one memory into the other memory. At the end of this process, the address recognizers in both units are disabled by clearing the *AP-Match* registers.
 - d. Clear the *Toggle-Master/Checker* bit in the *FRC* register in both units. This action allows the *Toggle-Primary/Shadow* bit in the *QMR* register to be correctly set later in the marriage sequence.
4. Synchronize the caches by sending an *Invalidate-Cache* command (see Appendix A for the description of the *Invalidate-Cache* command).
5. Send an error report of any type from any BXU. This operation synchronizes the deadlock timers of the BXU's in the *Primary-Elect* and *Shadow-Elect* units.
6. Prepare the *Shadow-Elect* unit for marriage by using Physical Address IAC's to perform the following actions:
 - a. Copy the contents of the registers in the *Primary-Elect* pair to the registers of the *Shadow-Elect* pair except for the *QMR*, *Spouse-ID*, *Module-Error-ID*, *Bus-Error-ID*, *Logical-ID*, and *Physical-ID* registers. Because the *Arbitration-ID* register is a write-only register, software must write to both registers with the same value.
 - b. Set the *Module-Error-ID* register in the *Shadow-Elect* unit to a unique ID.

- c. Set the *Bus-Error-ID* register in the *Shadow-Elect* unit by setting the *Source-ID* field to a unique ID.
7. Make units aware of each other by using Physical Address IAC's to perform the following actions:
 - a. Copy the *Confinement-ID* field in the *Module-Error-ID* register of the *Primary-Elect* unit to the *Spouse-ID* register of the *Shadow-Elect* unit.
 - b. Copy the *Confinement-ID* field in the *Module-Error-ID* register of the *Shadow-Elect* unit to the *Spouse-ID* register of the *Primary-Elect* unit.
8. Using IAC type 0100_B (Register Request Using a Physical IAC), copy the *Logical-ID* number of the *Primary-Elect* unit to the *Shadow-Elect* unit to synchronize the *Primary-Elect/Shadow-Elect* pair.
9. Marry the *Primary-Elect/Shadow-Elect* units, using IAC type 0010_B (Register Request Using a Logical IAC). By using this IAC type to access the *QMR* register, the same bits are simultaneously set in all BXUs. Marriage is accomplished by performing the following actions:
 - a. If toggling between *Primary-Elect* and the *Shadow-Elect* units is desired, set the *Toggle-Primary/Shadow* bit in the *QMR* register in both modules. Otherwise, this bit is cleared in the modules.
 - b. Set the *Married* bit in the *QMR* register of the *Shadow-Elect* unit and *Primary-Elect* unit. This marks the modules as married.
10. For an active/passive module, set up the married modules to recognize the same memory ranges by using Logical Address IAC's. The Logical Address IAC is used to simultaneously set up the *Primary-Elect* and the *Shadow-Elect* units.
Address recognition is accomplished by performing the following functions:
 - a. Set the *AP-Match* registers to the desired value.
 - b. Set the *AP-Mask* registers to the desired value.
 - c. Set the *Enable* bit in the *AP-Match* registers.
11. Activate the 80960MC processor(s) in the modules by sending an *Restart-Processor* IAC.

If either module in the FRC pair fails, all the BXU's in the failed FRC module deactivate themselves, and their spouses take over operation. This allows recovery to be transparent to the rest of the system. The recovery procedures are activated based on the error type and location information which was received by the BXU on the error report lines.

The following steps show how to divorce the primary/shadow unit in a married processor module:

1. Stop the processors in the married pair by sending a "Stop-Processor" IAC message.
2. Reset the *Married* bit in the *QMR* register.
3. Change the ID's in the *Arbitration-ID* register.

4. Change the ID's in the *Unit-ID* field of the *Logical-ID* register.
5. Start the processors of the resulting modules by sending a "Start-Processor" IAC message.

Processor Module Recovery

A primary/shadow pair automatically recovers from any single failure in the module confinement area. The recovery process for a permanent error in a module assumes that a primary and a shadow unit are available in the system. The shadow unit takes over if there is a permanent error in the primary unit, or the primary unit takes over if there is a permanent error in the shadow unit. An interrupt message can be sent to notify the operating system of the faulty module.

An error is considered to occur in this module if any of the following actions occur:

- The *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Module-Error-ID* register.
- A *Primary-Catastrophe* error report is received and the *Shadow* bit in the *QMR* register has a value of zero.
- A *Shadow-Catastrophe* error report is received and the *Shadow* bit in the *QMR* register has a value of one.

An error is considered to occur in this module's partner if any of the following actions takes place:

- The *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Spouse-ID* register.
- A *Primary-Catastrophe* error report is received and the *Shadow* bit in the *QMR* register has a value of one.
- A *Shadow-Catastrophe* error report is received and the *Shadow* bit in the *QMR* register has a value of zero.

Actions by the Failed Module

The failed module (FRC Pair) isolates itself from the AP-bus and ignores all requests from the L-bus. Requests from the AP-bus are ignored, except for IAC access type 0100_B (Register Request using a Physical Address) and IAC access type 0111_B (Identify Device IAC). The IAC requests using a physical address can be used by diagnostic software to probe into the failed module. If another permanent error is reported in this module, however, then the IAC access type 0100_B is also disabled.

On receiving an error report indicating a permanent error in this module, all BXUs in the module take the following actions:

- The BXUs set the *Faulty* bit in the *FT2* register (see Appendix A for the description of the *FT2* register). This disables the AP-bus recognizers for memory addresses, IAC messages (IAC

access type 0011_B), and IAC requests using a logical address (IAC access type 0010_B). The AP-bus recognizers for IAC requests using a physical address (access type 0100_B) and Identify Device IAC (access type 0111_B) remain enabled. The L-bus recognizers for all IAC and memory addresses are disabled, however. When the *Faulty* bit is set, the BXU also drives the *COM* pin low. Under this condition, the BXU can still generate error reports and will accept error reports from the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines, but it will not propagate error reports from the $\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$ lines to the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines.

- The BXUs reset the *Married* bit in the *QMR* register.
- If the *Faulty* bit was set before this error report was received, then the BXUs set the *Inactive* bit in the *FTI* register. This disables the AP-bus recognizers for the memory addresses, the IAC messages, and the IAC requests using physical or logical addresses (the Identify Device IAC requests are accepted). These actions totally isolate the module from the AP-bus. The BXU does not generate error reports, nor does it propagate error reports from the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines to the $\overline{\text{LERL}}_1$ - $\overline{\text{LERL}}_0$ lines. The BXU does not accept error reports from $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines unless the BXU is in Memory mode (the *BXU-Mode* bit in the *LBI-Control* register is zero). The *Inactive* bit in the *FTI* register is set conditionally to allow diagnostic software to have access to the BXU, if at all possible.

Actions by Partner Module

If a module's partner fails, the surviving module takes over operation. On receiving an error report indicating a permanent module error in the partner module, all BXUs in this module reset the *Married* bit in the *QMR* register. This causes this module to respond to the AP-bus requests.

Memory Module

A memory module consists of a BXU (in Memory mode), a memory controller, and a RAM array, as shown in Figure 13-4. This section describes the response to an error report, the ECC interface, memory module shadowing, and memory module marriages. Except for a correctable and uncorrectable array errors, memory module recovery is identical to processor module recovery, which was covered in the previous section.

Response to Error Reports

When an error report occurs, the data stops flowing through the BXU. If the BXU was in the process of driving a write request on the L-bus, then the remaining data cycles are invalid and the $\overline{\text{BE}}_3$ - $\overline{\text{BE}}_0$ lines are not asserted. During the transient waiting period, the entire write request is reissued on the L-bus. This ensures that the memory array holds valid data before retry begins. If the BXU was performing a read request at the time of the error report, the L-bus cycle is completed, but the data is not used.

A BXU in Memory mode sets all of its *RMW-Lock* bits in the *Lock* register on every error report. The lock timer begins at the end of the transient waiting period.

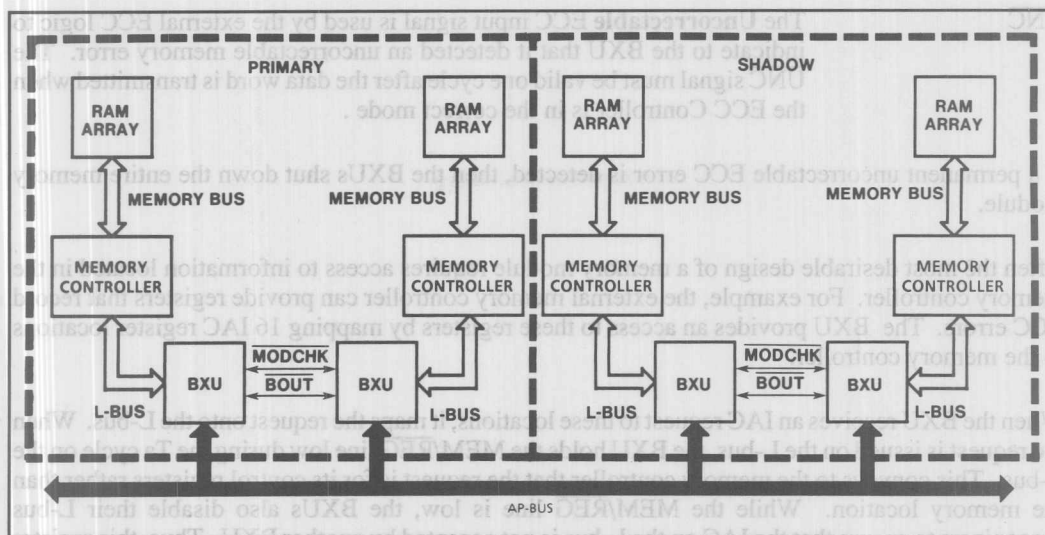


Figure 13-4: Memory Modules

Memory Controller Interface

The BXU supports discrete memory controllers that provide single error correcting and double error detecting ECC protection for the memory array. An ECC error signal from an external controller causes the BXU to assert the \overline{BERL}_1 - \overline{BERL}_0 lines with the same timing as the AP-bus parity bits. Thus, ECC errors are handled correctly even if the *BERL-Wait* bit in the *FTI* register is disabled (see the “ \overline{BERL}_1 - \overline{BERL}_0 Timing” section in Chapter 12. The interface between the BXU and the external ECC logic consists of three signals: Correct (\overline{COR}), ECC Error (ECC), and Uncorrectable ECC (\overline{UNC}).

\overline{COR}

The **Correct** signal is used by the BXU to inform the external memory controller to correct the memory data as it flows onto the L-bus. If this signal is not asserted, then the memory data may flow directly onto the L-bus with only error checking, but no correction. \overline{COR} is asserted at least one cycle before the next memory address is sent on the L-bus. If \overline{COR} is asserted, then the *Fast-Reply* bit in the *Lock* register is over-ridden, and the data is buffered before going on the AP-bus.

\overline{ECC}

The **ECC Error** input signal indicates that an error was detected from the external ECC logic. \overline{ECC} is asserted if the ECC logic detected an error in the memory data. This signal may be asserted even though the external logic may be correcting the error and providing correct data on the L-bus. If the BXU is asserting its \overline{COR} signal, the \overline{ECC} signal is ignored. Only the \overline{UNC} pin is checked for an error indication under these conditions. The \overline{ECC} signal must be valid one cycle after the data word is transmitted.

UNC

The **Uncorrectable** ECC input signal is used by the external ECC logic to indicate to the BXU that it detected an uncorrectable memory error. The **UNC** signal must be valid one cycle after the data word is transmitted when the ECC Controller is in the correct mode.

If a permanent uncorrectable ECC error is detected, then the BXUs shut down the entire memory module.

Often the most desirable design of a memory module requires access to information located in the memory controller. For example, the external memory controller can provide registers that record ECC errors. The BXU provides an access to these registers by mapping 16 IAC register locations to the memory controller.

When the BXU receives an IAC request to these locations, it maps the request onto the L-bus. When the request is issued on the L-bus, the BXU holds the MEM/REG line low during the T_a cycle on the L-bus. This conveys to the memory controller that the request is for its control registers rather than the memory location. While the MEM/REG line is low, the BXUs also disable their L-bus recognizers to ensure that the IAC on the L-bus is not accepted by another BXU. Thus, this register access facility, along with the **COM** pin, and the **COM** register of the BXU can assist software to initialize the discrete memory controller.

Restoring Failed Memory Locations

The external memory controller must provide a mechanism for restoring correct values to memory locations that have single bit errors. This could be a scrubbing mechanism during the refresh cycles or it could be a special diagnostic memory access. The only constraint is that it must be done atomically (read, correct, and restore without interruption) and in lockstep even though only one of the modules may have an error.

Failures in Partial Write Operations

If a partial write access encounters an address location that has an uncorrectable error, it must leave the location in a state that causes subsequent read accesses to generate an uncorrectable error. Any other words that are written as part of the partial write operation must be written completely. At the completion of the partial write operation, all locations must have their new data values except the faulty location, which returns an uncorrectable error on any subsequent read operations to the location.

Note that this means that the controller cannot assert either the **UNC** or the **ECC** signals. Asserting these signals generates an error report. Because the BXU issues the AP-bus reply before the write operation has completed, there is no guarantee that the request will be retried. Thus, the memory could remain in an incorrect state. If the memory controller wants to abort this partial write operation, it must assert the **ERR** signal. This causes the BXU to generate an **Unsafe-Confinement-Area** error report that immediately shuts down the entire module.

memory module shadowing

Most of the facilities required to support memory module shadowing are the same as for processor module shadowing. The only additional requirement is that the refresh function of the two modules must operate in lockstep.

To accomplish this, the BXU uses the Force Refresh ($\overline{\text{FRF}}$) signal that is activated by the *Sync-Refresh* command (see Appendix A for the description of the *Sync-Refresh* command). The $\overline{\text{FRF}}$ signal is an output from the BXU that informs the external memory controller to immediately perform a refresh function.

When the BXU receives a *Sync-Refresh* command, it generates an error report with error type *Sync-Refresh*. If the BXU receives a *Sync-Refresh* error report and if the *Confinement-ID* field matches either the *Confinement-ID* field in the *Module-Error-ID* register or the *Spouse-ID* register, then the BXU asserts the $\overline{\text{FRF}}$ pin for one cycle. In other words, receiving a *Sync-Refresh* error type with its own or its spouse's module identification causes the BXU to assert the $\overline{\text{FRF}}$ signal for one cycle at the beginning of the transient waiting period.

The external memory controller uses the $\overline{\text{FRF}}$ signal to force one or more refresh cycles in such a way that the two modules have refresh cycles operating in lockstep. An error report is used because all traffic to the modules is stopped. Thus, the refresh commands have exactly the same timing in both modules. If this error report is received twice in a row, however, it would be handled as a normal permanent error (shutting down the module).

Bringing up a memory module shadow when one of the modules is actively providing data to the system places restrictions on system operation. The only way to copy the data is to perform standard memory read and write operations.

Memory Module Marriage

The following steps, which are similar to those of the processor module, need to be taken to marry two memory modules:

- Place the module (components and logic on the L-bus) in the same state.
- Place the BXUs in the same state.
- Make the primary and shadow units aware of each other by setting their *Spouse-ID* register equal to their respective *Module-Error-ID* register.
- Marry the primary and shadow units by setting the *QMR* register.
- Cause an event that synchronizes the primary and shadow units, such as sending a *Sync-Refresh* command.

The memory refresh synchronization of the primary and shadow units is the basic difference between marrying memory modules and marrying processor modules. The *Sync-Refresh* command causes the external memory controller to be synchronized to the system clock. When using this command, the *Toggle-Primary/Shadow* bit in the *QMR* register must first be cleared. After using this command and resetting the *Silent* bit in the *QMR* register, the *Toggle-Primary/Shadow* bit may be set.

Bus Switching

To recover from permanent AP-bus errors, at least one additional AP-bus must be available. Figure 13-5 illustrates a dual-bus *Primary/Shadow* pair. Note that each module is attached to two buses (i.e., it is dual ported). From the viewpoint of a given BXU, the system bus that it is attached to is its *Primary* bus, and the other bus is its backup.

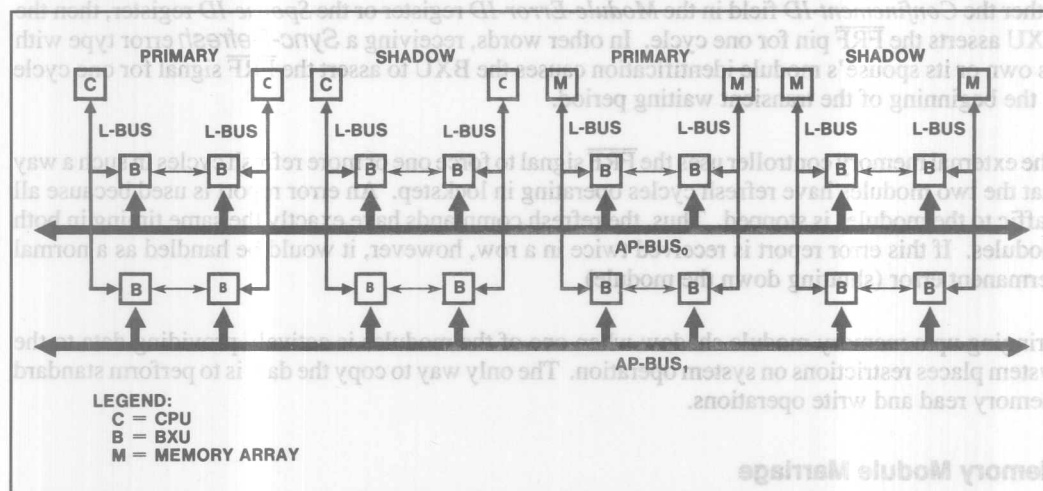


Figure 13-5: *Primary/Shadow* BXUs with Two AP Buses

The BXUs in a system are capable of automatically switching AP-buses when a permanent error causes one AP-bus to stop functioning. When a permanent error is detected, all the BXUs on the faulty AP-bus isolate their modules from this bus. The BXU on the failed AP-bus ignores L-bus requests, while the BXU on the backup AP-bus picks up these requests. The data that normally would have flowed on the failed bus, now flows on the backup bus. If a BXU has a cache, the BXU invalidates its cache directory because the directory must be reorganized to match the new (and larger) address space, including a new interleaving factor.

During normal operation, all AP-buses in the system may be used to transmit data because the AP-bus does not hold state information. AP-buses do not operate in lock-step (FRC), thus the data on each individual bus are unique packets. This multiplicity of AP-buses increases the system throughput without sacrificing back-up capability. Normally the memory address range is interleaved between the two buses, but this is not required for bus switching.

While processor modules are connected to all active AP-buses, a memory module connects only to the AP-bus with the corresponding address range. It also connects to the backup bus, although this connection is normally inactive. Memory-mapped I/O modules, in general, connect to all buses in interleaved systems. Should the *primary* bus fail, the BXUs that connect the memory module to the failed bus isolate themselves from the failed bus, and the BXUs on the backup bus become active and begin receiving requests. Figure 13-6 shows an example of bus switching.

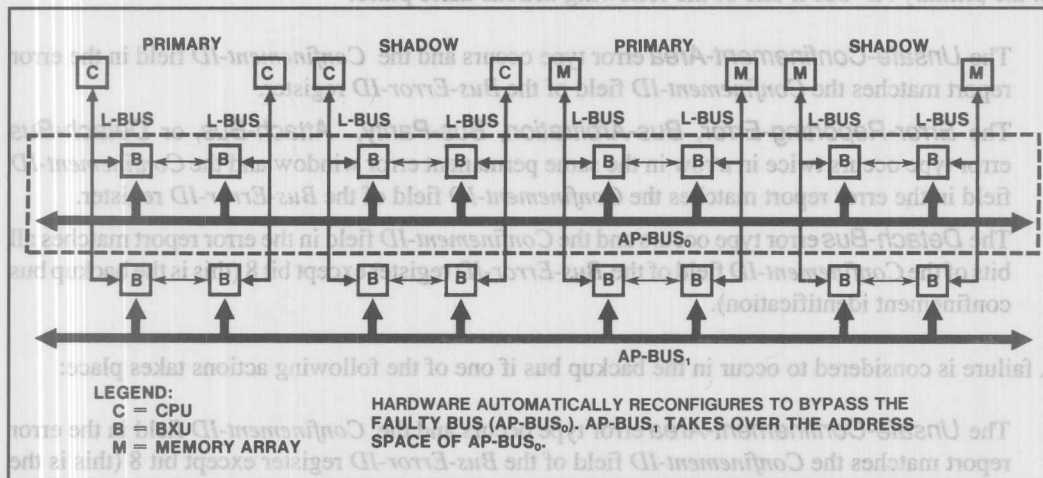


Figure 13-6: Permanent Bus Error Recovery

The only bus pairings that are allowed are AP-bus₀ and AP-bus₁, and AP-bus₂ and AP-bus₃. This restriction is required because recovery occurs by changing the interleaving factor. Because of this restriction, there is no need to make the BXUs aware of their backup bus (they always know the one and only bus that can backup their bus). Setting up a bus pair is done with the following steps. For each step, the BXUs on both buses must be setup.

1. The *Function* bits in the *Match* registers must be set to a compatible set of values (see Appendix A for more details on the *Function* bits).
2. The *Bus-Switching-Disable* bit in the *AP-Control* register must be cleared.

Clearing the *Bus-Switch-Disable* bit in the *AP-Control* register allows a special use of RPYDEF. The RPYDEF signal is used only to disable the bus time-out. No reordering of the bus pipeline occurs.

Bus switching provides recovery for memory access and IAC Messages. IAC register requests that are addressed to BXUs on the failed bus will return bad-access replies (because the request will have been routed out on the backup bus). The BXUs on the failed bus are still accessible to the processors on their L-bus by using IAC requests.

The remaining parts of this section describe the details of the bus switching mechanism and present the system implications of using this mechanism.

Bus Recovery

When an AP-bus fails, all traffic is routed to the backup bus for a system with paired AP-buses. This section describes the details of how this is done.

An error occurs either on the primary AP-bus or the backup AP-bus. A failure is considered to occur on the primary AP-bus if one of the following actions takes place:

- The *Unsafe-Confinement-Area* error type occurs and the *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Bus-Error-ID* register.
- The *Error-Reporting-Error*, *Bus-Arbitration*, *Bus-Parity*, *Attach-Bus*, or *Detach-Bus* error type occurs twice in a row in the same permanent error window and the *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Bus-Error-ID* register.
- The *Detach-Bus* error type occurs and the *Confinement-ID* field in the error report matches all bits of the *Confinement-ID* field of the *Bus-Error-ID* register except bit 8 (this is the backup bus confinement identification).

A failure is considered to occur in the backup bus if one of the following actions takes place:

- The *Unsafe-Confinement-Area* error type occurs and the *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Bus-Error-ID* register except bit 8 (this is the backup bus confinement identification).
- The *Error-Reporting-Error*, *Bus-Arbitration*, *Bus-Parity*, *Attach-Bus*, or *Detach-Bus* error type occurs twice in a row in the same permanent error window and the *Confinement-ID* field in the error report matches the *Confinement-ID* field of the *Bus-Error-ID* register except bit 8 (this is the backup bus confinement identification).
- The *Detach-Bus* error type occurs and the *Confinement-ID* field in the error report matches all bits of the *Confinement-ID* field of the *Bus-Error-ID* register.

Actions by the Failed Bus

All BXUs on the failed AP-bus isolate their modules from the faulty bus. Consequently, requests from the faulty bus are ignored. L-bus requests for this AP-bus are ignored by the failed BXU, and are picked up by its backup. The failed BXU, however, accepts its L-bus IAC requests and responds to them.

On receiving an error report indicating a permanent bus error on this AP-bus, all BXUs on this bus take the following actions:

- The BXUs set the *Inactive* bit in the *FTI* register. This disables the AP-bus recognizers for memory addresses, IAC message (IAC access type 0011_b), IAC requests using a logical address (IAC access type 0010_b), and IAC requests using a physical address (IAC access type 0100_b). The BXUs still recognize the Identify Device IAC (access type 0111_b).

- The BXU does not accept error reports from the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines unless the BXU is in Memory mode. The L-bus recognizers of the BXU are disabled for memory addresses. The BXU accepts only IAC requests addressed to it. No IAC requests are accepted that are for another BXU on the failed bus, however. The BXU stops tracking and buffering requests handled by its partner, and begins tracking and buffering requests from the L-bus that would normally have been routed through this BXU (but are now being handled by the backup BXU).
- The BXUs invalidate their cache directory.
- The BXUs clear the *Enable*, *I/O-Channel₀-Active*, and *I/O-Channel₁-Active* bits in the *Prefetch-Control* register to disable the I/O prefetching function.

Actions by the Backup Bus

All BXUs on the backup bus pick up the traffic that normally would have flowed onto the failed bus.

Upon receiving an error report that indicates a permanent error in the partner's bus, all BXUs on this bus take the following actions:

- The BXUs set the *Backup-Bus-Inactive* bit in the *FT2* register. This causes several actions to occur:
 - The BXU retries all requests that it buffered internally, not just its own requests.
 - Bus requests that were pending at the time of the error are sent on this AP-bus.
 - The L-bus memory address recognizers ignore LAD_4 if they were interleaved, or become active if they were non-interleaved and considered a backup (see Appendix A for the definition of the *Function* bits of the *Match* and *Mask* registers).
 - The IAC address recognizers match on all requests that would have flowed onto the failed AP-bus.
 - The new interleaving information is sent to the cache logic to update its address mapping logic.
- The BXUs invalidate the cache directory. Because this directory now handles a larger address space, the cache directory is reorganized to match the new address space (for instance a different interleaving factor). Before reorganizing the directory, the old one is invalidated. Because the cache uses a write-through update policy, this invalidation can be done simply in the BXU. There is no need for additional updates to memory.
- The BXU clears the *Enable*, *I/O-Channel₀-Active*, and *I/O-Channel₁-Active* bit in the *Prefetch-Control* register. This disables the I/O prefetch function. This action prevents errors due to the potential change in interleaving factor (the data returned from the prefetch buffers might be the incorrect data). All requests that normally would have been handled by the prefetch unit are handled as a normal AP-bus memory request.

Requests to a Failed Bus

Requests to a failed AP-bus can result in one of several replies listed below:

- If the request is to the BXU on the processor's L-bus, then the BXU processes the request and replies correctly.
- If the request is to a BXU in another module, then the request (either memory or IAC) is sent to the backup AP-bus (assuming that a backup AP-bus exists). This action results in a normal reply for a memory request or a message IAC (access type 0011_B). For an IAC requests that uses a physical or logical address (access types 0100_B or 0010_B), the bad-access reply is used because the IAC address specifies a BXU on a failed bus.
- If the request is to a BXU in another module and there is no backup AP-bus, then the request suspends activity on the processor's L-bus. Because no BXU on the L-bus recognizes the request, no BXU asserts the READY signal.

IAC Operations After a Bus Switch

After a bus failure and bus switch, IAC register requests that normally would have used the failed bus, are routed to the backup bus. These requests return bad-access replies because no BXU on the backup bus matches the *Bus-Destination* field in the IAC address. The BXUs on the failed bus may still be accessed from their L-buses. IAC requests to the local BXUs are not mapped to the backup bus.

If the failed bus was the message bus (AP-bus₀), then the backup bus (AP-bus₁) takes over as the message bus. All IAC messages are now handled over the new bus. This switch is transparent to the 80960MC processor. The backup bus (AP-bus₁) can also become the message bus if a *Detach-Bus* command is issued for the message bus confinement area. If this command occurs, an *Attach-Bus* command can be issued for the original message bus confinement area making AP-bus₀ the message bus again.

The next few paragraphs show the details of how the message bus switching operates. BXUs that are not currently the message BXUs track the actions of the message BXUs in the following way.

- Write operations to the *Processor-Priority* register are done in all BXUs on the L-bus using the "Register Request From the L-Bus" IAC (access type 0000). Consequently, all *Processor-Priority* registers always have the most recent priority information (see Appendix A for the description of the *Processor-Priority* register).
- The *Message-Buffer-Full* bits in the *Processor-Priority* register track the status of the $\overline{\text{IAC}}$ lines. If $\overline{\text{IAC}}$ is asserted, then the appropriate *Message-Buffer-Full* bit is set in all non-message BXUs. If $\overline{\text{IAC}}$ is not asserted, then the appropriate *Message-Buffer-Full* bit is cleared. Thus, the non-message BXUs always contain the most recent status of the message buffer registers.
- The *Message-Data-Valid* bit in the *Processor-Priority* register remains cleared in the non-message BXUs.

As a result of a bus switch, the backup bus BXU becomes the message BXU. Because the backup BXU was tracking the message state information, it is prepared to immediately respond to IAC message requests from the AP-bus. If an IAC message was waiting for the 80960MC processor before the bus switch, then the *Message-Data-Valid* bit in the original message BXU is set, and the *Message-Buffer-Full* bit in all the BXUs are set. When a message-buffer read request is issued, the BXU with the associated *Message-Data-Valid* bit set replies with the data. Thus, if a bus switch occurs after an IAC message was received, but before the 80960MC processor read the message buffer, the IAC message is correctly passed on to the 80960MC processor.

Attach-Bus Command

The *Attach-Bus* and *Detach-Bus* commands can be used to switch the message bus or to perform diagnostics on the AP-buses. For example, if the 80960MC processor issues a *Detach-Bus* command, the BXU responds by issuing a *Detach-Bus* error report, which removes the bus from operation and causes the backup bus to handle all the requests. After tests verify correct operation of bus switching, the *Attach-Bus* command connects the AP-bus back to the system. The BXU responds by sending an *Attach-Bus* error report.

The *Attach-Bus* error report establishes operation on a pair of buses again. The following sequence shows how to attach a bus:

- Set the *Sequence* bit in the *Match* register of every BXU on the pair of functional buses. (The *Sequence* bit in the *Match* registers of the BXUs on the other bus pair of a four bus system do not need to be set.) The *Sequence* bit is set to ensure that there is no more than one access pending from the L-bus to this AP-bus pair.
- Clear the *Enable*, *I/O-Channel₀-Active*, and *I/O-Channel₁-Active* bits in the *Prefetch-Control* register in every BXU on the pair of functional buses. This prevents the prefetch unit from generating additional accesses on the bus that might prevent the correct switching.
- Wait approximately 100 cycles after the last *Sequence* bit is set. This action ensures that all modules are operating in sequential mode.
- Send an *Attach-Bus* command to one BXU on the functional bus of the pair. This command causes an *Attach-Bus* error report to be generated with the *Location-ID* field equal to the *Bus-Error-ID* register of that BXU (the *Confinement-ID* field will match the *Bus-Error-ID* register of the currently operating bus in the pair).

The response to the *Attach-Bus* command depends on whether the bus or its backup bus was removed. If the *Confinement-ID* field matches the *Bus-Error-ID* register, then clear the *Backup-Bus-Inactive* bit in the *FT2* register. This action returns the L-bus address recognizers to their normal state. If the *Confinement-ID* field matches the *Bus-Error-ID* register except for bit 8 (i.e., the backup bus), then clear the *Inactive* bit in the *FT1* register. This action returns the L-bus address recognizers to their normal state.

Both BXUs in the bus pair in each module re-evaluate the address of the one request that may be outstanding in the BXUs and decide whether they should retry the request. For an outstanding read

operation, the BXU finishes the request during retry on the AP-bus that normally handled this operation when two AP-buses were available. For an outstanding write operation, the BXU finishes the request during retry on the AP-bus that handled the request when only one AP-bus was available (even if two AP-buses are available now).

If an *Attach-Bus* error report is received twice in a row, it is labeled a permanent error, and the normal permanent error operations will occur (i.e., the bus is removed from service since this report uses the *Bus-Error-ID* as its location identification).

Communication Between Buses

When a BXU determines that its backup bus has failed, it retries any outstanding requests of the BXU attached to the failed bus, as well as its own requests. As with retry sequence without bus switching, the requests from one module are issued in the same order as they were received on the L-bus. The internal queuing mechanism in the BXU allows it to combine the requests from the two buses in the correct order.

The BXUs within a module use two signals, POPQUE and Subsystem Busy (SSBUSY), to coordinate the activity between the AP-buses. Figure 13-7 shows how the POPQUE and the SSBUSY signals are connected.

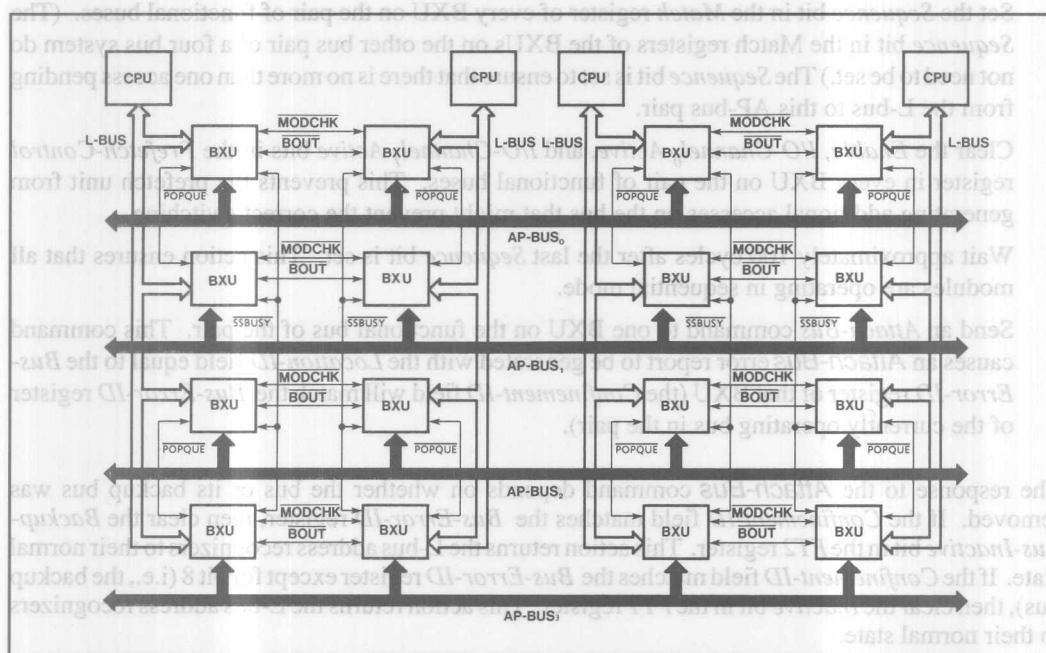


Figure 13-7: System Connection to POPQUE and SSBUSY

The pair of BXUs use the $\overline{\text{POPQUE}}$ signal to inform each other of the state of pending requests on their AP-bus. Since read requests and IAC requests hold up the L-bus until the data is returned, only write memory requests are monitored. Although only one BXU actually performs the write operation, the write requests are buffered in both BXUs associated with the paired AP-buses. When the operation is completed on the AP-bus, the partner BXU is informed by the $\overline{\text{POPQUE}}$ pin. This signal is asserted when the write request that has been in the pipeline queue the longest time is completed.

The internal queuing and $\overline{\text{POPQUE}}$ signaling protocol ensures that every request issued by the 80960MC processor on the L-bus is serviced once and only once by the addressed component on the AP-bus. During the transient wait period a different protocol conveys the status of all write requests.

The $\overline{\text{SSBUSY}}$ signal connects all the BXUs in a module (that are in the same subsystem). When this signal is asserted (low), the BXUs accept the address of the request, but do not transfer data. This signal ensures that the BXUs handle RMW-Write requests, IAC requests, and retry operations correctly. The $\overline{\text{SSBUSY}}$ signal ensures that the all BXUs are not busy handling IAC requests or performing prefetch operations.

Memory Range Recognition Considerations

The BXU provides address recognizers that can be individually programmed for interleaved or non-interleaved accesses to the AP-bus. The simplified diagram of Figure 13-8 illustrates how the BXU effects recovery for three cases: range recognition only (no interleaving), bus interleaving only, and both range recognition and bus interleaving.

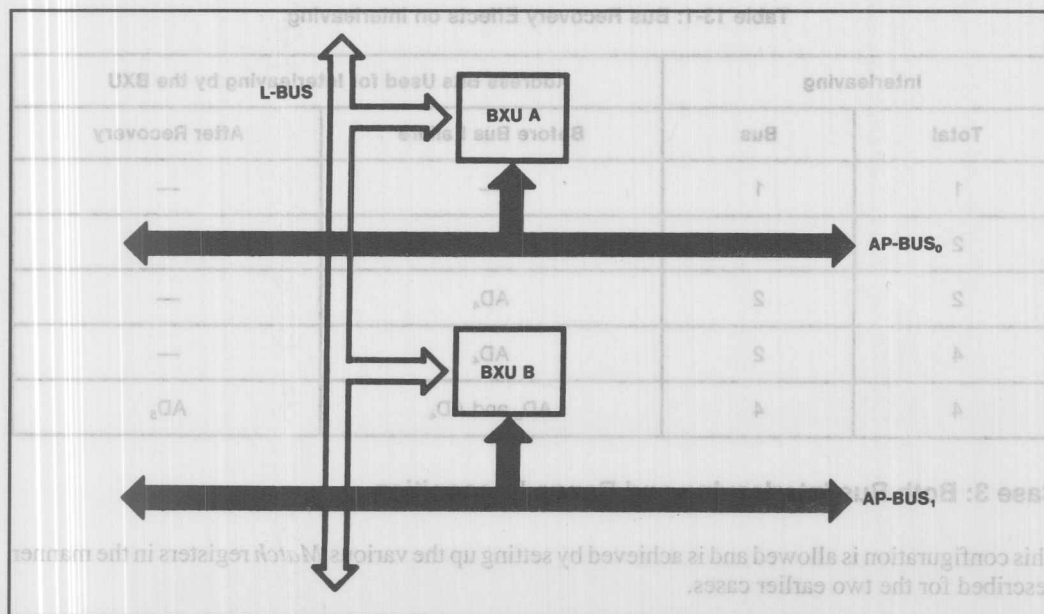


Figure 13-8: Bus Recovery Illustration

Case 1: Range Recognition Only (No Interleaving)

The address recognizers in BXUs A and B are set for no interleaving. To recover from a permanent bus error on AP-bus₀, the following must be true: for every *Match* register in BXU A that has its *Function* bits set to 10_B, the corresponding *Match* register in BXU B must cover the same range and have its *Function* bits set to 11_B (and vice versa).

In normal operation requests only flow through BXU A. In case of a bus switch, the address recognizers of BXU B are activated and begin to pick up the traffic that was previously flowing through BXU A.

Case 2: Bus Interleaving Only

The address recognizers in BXU A and BXU B are set to interleave between AP-bus₀ and AP-bus₁. A permanent bus error on AP-bus₀ is recoverable under the following conditions: the *Function* bits in all *Match* registers are set to 01_B, and the two BXUs have complementary interleaving factors set up in the *LBI-Control* register (the *Mask* bits should be the same and the *Match* bits should be the inverse of each other in the low order bit).

Recovery is accomplished by changing the interleaving factor. This change in interleaving affects the L-bus address recognizers and forces the BXU to recognize different address bits after recovery. Table 13-1 illustrates the address bits that the BXU interleaves before and after a bus error forces a bus switch. Note that the total memory interleaving (i.e., the combination of bus and memory controller interleaving) remains constant through the recovery sequence for any of the modes.

Table 13-1: Bus Recovery Effects on Interleaving

Interleaving		Address Bits Used for Interleaving by the BXU	
Total	Bus	Before Bus Failure	After Recovery
1	1	—	—
2	1	—	—
2	2	AD ₄	—
4	2	AD ₄	—
4	4	AD ₅ and AD ₄	AD ₅

Case 3: Both Bus Interleaving and Range Recognition

This configuration is allowed and is achieved by setting up the various *Match* registers in the manner described for the two earlier cases.

Memory Considerations

The bus switch for a memory module is the same as for processor modules, plus one added operation. The BXU on the surviving bus of the bus pair sets all of its *RMW-Lock* bits in the *Lock* register. This action ensures that any *RMW-Lock* bit that was set before the bus failed is honored correctly. Although some *RMW-Lock* bits may be set unnecessarily, these extra bits are cleared by the lock time-out.

Cache Considerations

As a result of a bus switch (permanent bus error, *Detach-Bus* command, or *Attach-Bus* command), all BXUs on the paired AP-buses invalidate their entire cache directories and set the *Line-Configuration* bit in the *Cache-Configuration* register (forcing eight lines per block, which is the only non-interleaved configuration that is supported). This invalidation is necessary to prevent incorrect mapping of data in the cache.

The remaining BXUs establish their directories again using the new configuration. Table 13-2 shows the change in the cache configuration as a result of bus switching actions (see the *Cache-Configuration* register described in Appendix A for definition of configuration numbers). The cache remains enabled, and the memory requests that follow the bus switch refill the cache. In a four AP-bus configuration, the BXUs attached to the other paired bus do not invalidate their cache configuration.

Table 13-2: Cache State Change Caused by Bus Switch

Configuration Before Failure or After <i>Attach-Bus</i> Command	Configuration After Failure or After <i>Detach-Bus</i> Command or Before <i>Attach-Bus</i> Command
<div> <div>Failure or <i>Detach-Bus</i> Command</div> <div>Attach-Bus Command</div> </div>	
Four-way interleaved (LAD ₄ and LAD ₅), configuration #1 or #4.	Two-way interleaved (LAD ₅)
Two-way interleaved (LAD ₄), configuration #2 or #5.	No interleaving, configuration #3
No interleaving, configuration #3	Off

NOTES:

1. The configuration numbers are from the description of the *Cache-Configuration* register.
2. These changes are true only for the surviving bus in the bus pair. If there is another pair of buses (i.e., 4 buses total), the other BXUs do not have any configuration change.

In four-way interleaved systems, the cache size is reduced to five-eighths of its original size (four-eighths available for the operational bus in the pair and one-eighth available for the surviving bus in

the failed pair). Although the size of the cache is reduced, all SRAM locations are used. This sizing occurs because of the double mapping of the address bits (some bits are used by the cache directory as *tag* bits and by the SRAM as address bits).

In a two-way interleaved cache, the cache size is reduced to one-half of its original size.

I/O Prefetch Considerations

A bus switch affects BXUs that perform I/O prefetch functions. When a bus switch occurs, both prefetch channels in the two BXUs (on the failed and surviving AP-bus) are disabled by resetting the *Enable* bit in the *Prefetch-Control* register. This is done because the changes in interleaving factors may invalidate the data buffers of the prefetch unit. Data requests are still correctly serviced. However, the requests flow to AP-bus memory rather than being serviced from the prefetch buffer. Thus, the access time of the requests are slower as a result of the bus switch.

Before the prefetch channels can be used again, software must set the *Enable* bit. If a *Start* command is issued while the prefetch *Enable* bit is zero, the command is treated as a no operation.

After a bus switch, outstanding prefetch read requests are lost by the BXU that connects to a failed AP-bus. Consequently, the processor requesting the prefetched data experiences a longer latency for these read requests with address locations covered by the quiescent BXU. This condition will exist until the end of that transfer, but the re-initialization of either prefetch unit in the other BXU before the start of the next transfer will bring operation back to normal.

FRC Splitting

FRC splitting is a function provided by the BXU to allow systems to recover after a fault when module shadowing is not used. It allows recovery by splitting a master/checker pair so that one component continues to run while the other shuts down. FRC splitting can be used in applications that need enhanced reliability, but where the consequences of a fault are not severe. FRC splitting minimizes the number of resources required to achieve fault tolerance, but recovery is not transparent to software, proper identification of the faulty module is not ensured, and the system is not fault-tolerant after a FRC split. FRC splitting requires a single arbitration network shown in Figure 11-4.

FRC splitting occurs only with a RESET sequence. This form of recovery is enabled in all BXUs after a *cold* RESET. It may be disabled by software by using the *FRC-Splitting-Disable* bit in the *FRC-Splitting-Control* register. Some methods to reset an 80960 system are listed below.

- A software command can prompt external support logic to send a local RESET
- A watchdog timer can start the RESET sequence when the system fails
- A person can manually reset the system by pushing a button

This form of recovery is not a result of the error report. FRC splitting occurs as part of a *warm* RESET sequence. Consequently, this recovery approach is not transparent to the user or the software system.

Because FRC splitting eliminates the FRC error checking in the module, diagnostic software plays a role in the trial and error sequence that determines which half of the module is operating correctly. FRC splitting occurs if the following conditions exist:

- A **warm** RESET occurs
- *FRC-Splitting-Disable* in the *FRC-Splitting-Control* register is cleared
- *My-Permanent-Error* bit in the *FRC-Splitting-Control* register is set

The *My-Permanent-Error* bit in the *FRC-Splitting-Control* register is set if the *Confinement-ID* field in the error report matches the *Confinement-ID* field in the *Module-Error-ID* register, and the *Married* bit in the *QMR* register has a value of zero.

The BXUs in the FRC pair respond to the permanent error in the same manner as described “Processor Module Recovery” section. The faulty module is removed from the system, but the BXUs in the faulty module can execute FRC splitting on the next **warm** RESET, if that form of recovery is enabled.

If FRC splitting occurs, the *Separated-Master/Checker* bit in the *FRC-Splitting-Control* register is set. The B XU selects either the master or checker as the active component. When FRC splitting first occurs, the master is the active component, on a subsequent **warm** RESET the components toggle the active state. For example, the checker will be active on the second, fourth, etc. assertion of RESET and the master is active on the third, fifth, etc. assertion of RESET. The state of the two BXUs is shown Table 13-3.

Table 13-3: FRC Splitting Control Bits State

Control Bit (Register)	Original State		State on First, Third,...Reset		State on Second, Fourth,...Reset	
	Master	Checker	Master	Checker	Master	Checker
Active-Component	x	x	1	0	0	1
Master (FRC)	1	0	1	0	1	0
Toggle-M-C (FRC)	x	x	0	0	0	0
Passive (FSC)*	0	0	0	1	1	0
Separated-M-C (FSC)*	0	0	1	1	1	1

NOTE:

* FSC is the abbreviation for *FRC-Splitting-Control* register.

The setting of the *Passive* bit of the *FRC-Splitting-Control* prevents the passive component from driving the AP-bus or the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines. Since the passive component responds to IAC requests addressed to it, software can reunite a split pair.

For FRC splitting to function properly, duplication of the AP-bus arbitration lines is not allowed. Both the master and checker must be connected to the same set of arbitration lines, so that either the master or the checker can act as the sole active component for the module. If the arbitration lines were duplicated, then the two sets of arbitration lines would always be different after an FRC split.

SOFTWARE CONSIDERATIONS

Software performs two key recovery management functions: establishing the reconfiguration policy, and reviewing the recovery decisions made by the hardware. When a permanent error occurs, the hardware automatically removes the faulty module or bus in a QMR system. Software, however, can decide if the configuration is optimal, given the reduced resources available to the system.

For example, consider the system shown in Figure 13-9. Assume that the primary processor module failed. The system hardware detects the error, reports the error, and replaces the primary unit with the shadow unit. The processor can be notified of the error condition if a BERL or a LERL line is connected to one of its interrupt input lines. The resulting system configuration can detect an error, but it cannot recover from a failure if a failure should occur in this module without a partner. Software needs to decide whether this module is to be remarried or be allowed to operate without a partner.

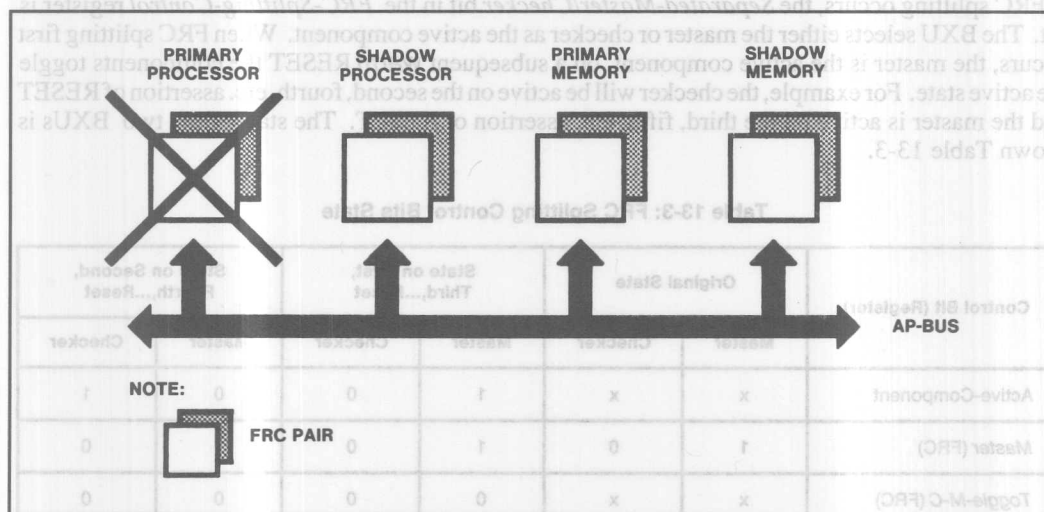


Figure 13-9: System Vulnerability After Reconfiguration

Software must also respond to transient errors by sending a *Terminate-Permanent-Error-Window* command after notification of this error by a processor. If the same transient error occurs twice in a row before the *Terminate-Permanent-Error-Window* command is issued, the transient error is classified as a permanent error (see the “Permanent Error Decision” for more details).

SUMMARY

The 80960 hardware system recovers from transient errors by using a retry sequence. The retry sequence provides recovery for most frequent failure modes (RAM soft errors, noise on the backplane, etc.). If the retry sequence fails, then it is possible to use a redundant resource to mask the fault from the rest of the system. Resource reconfiguration can successfully mask any single failure in the system. Resource reconfiguration requires redundant resources to use the recovery mechanisms, such as module shadowing, bus switching, or FRC splitting. The entire recovery algorithm is implemented in the BXU, and is orthogonal to both detection and redundancy mechanisms.

CHAPTER 14 INITIALIZATION

This chapter describes the hardware requirements for initializing the 80960MC processor and the BXU in a fault-tolerant system design. The basic minimum initialization requirements are described, as well as some of the available options. The exact initialization procedure depends on the type of system configuration. For example, the system can be configured to support multiprocessors, various levels of fault tolerance, up to four AP-buses, up to 20 modules, etc.

This chapter focuses on the initialization of a system that uses the BXU in multiprocessor and fault-tolerant configurations. The initialization procedure used for a single 80960MC processor configuration is explained in Chapter 3.

BXU INITIALIZATION

This section describes how to initialize the BXUs in the system. Initialization of the BXU is divided into three phases:

1. During the **default initialization** phase the default values are established in the BXU registers, the BXUs are synchronized to the clock phase, and the local bus identification parameters are established.
2. During the **identification** phase, the physical identification parameters of the BXU are established, and the board characteristics (type of board, amount of memory, etc.) may be loaded into the BXU.
3. During **parameterization** phase, the registers of the BXUs are loaded with any parameters that differ from the default values. During this phase, the logical identification parameters are set in the appropriate registers.

The default initialization phase is always required, regardless of the system configuration. The identification phase is required if the system configuration contains more than one BXU. The parameterization phase is optional and requires a 80960MC processor or another agent to perform the operations.

Default Initialization Phase

When the RESET signal is asserted, the BXU responds with the following actions:

- Forcing all internal state machines to the idle state.
- Synchronizing its clock phase to the global system clock phase.
- Loading its registers with default values.

Clock Phase Synchronization and RESET Timing

The assertion of RESET synchronizes the BXUs to a global system clock phase. Figure 14-1 illustrates the timing of RESET. RESET is asserted relative to any rising edge of CLK2, and held asserted for at least 44 CLK2 cycles. RESET then is deasserted after the rising edges of CLK and CLK2, but before the next rising edge of CLK2. This establishes clock edge A.

Because the fault-tolerant logic operates at half the AP-bus clock speed, the BXUs must be synchronized not only to the A edge, but to every other A edge. This implies that in a system operating at 16 MHz, the component clock is 32 MHz and the synchronizing clock for RESET at 8MHz (shown as CLK0.5 in Figure 14-1).

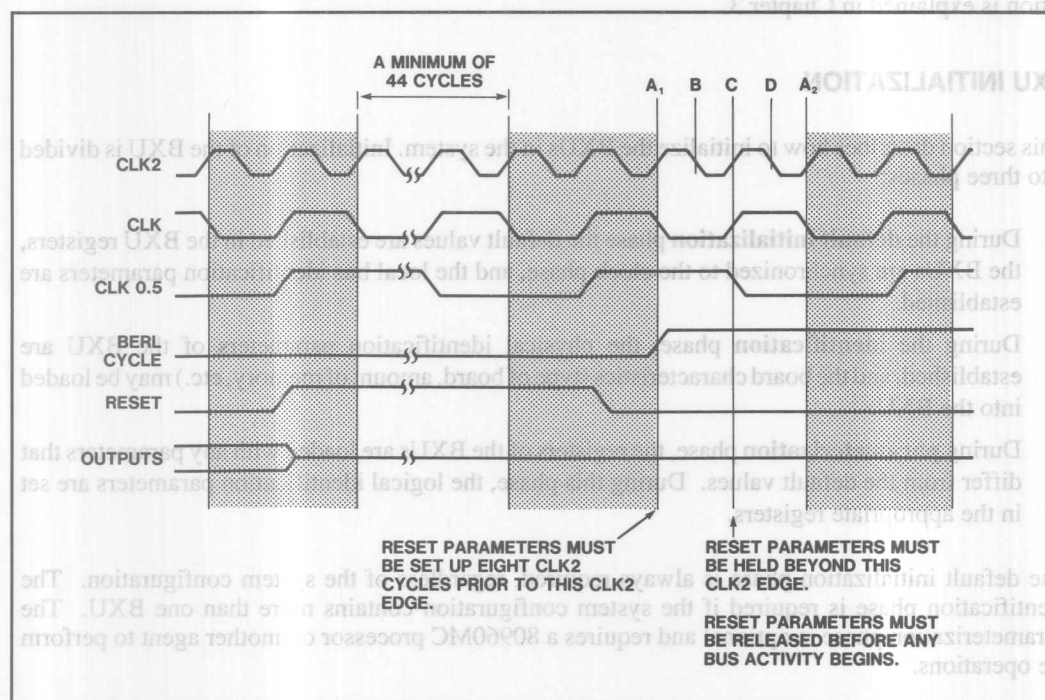


Figure 14-1: RESET Timing

The initialization parameters must be setup prior to the rising edge of CLK2, as shown in 14-1, and must be held beyond the first C edge. The normal setup and hold time specifications apply.

The outputs are placed in high impedance or are not asserted (for open-drain outputs) starting at the CLK2 edge at which RESET (asserted) is sampled.

All registers of the BXU except the *Error-Record* and the *FRC-Splitting-Control* registers are loaded in the same manner. While RESET is asserted, the register bits are either loaded with the default value

or the value sampled on an external set of pins. The final value loaded into the register is determined by the pins values on the last cycle that RESET was asserted.

The contents of the *Error-Record* register are not modified by the initialization sequence. The information in this register may be useful in determining the cause of a system crash. Thus, it is important for software to write to this register before reading it, unless software knows that the last RESET cycle was a *warm*RESET (see the “Cold and Warm Reset Distinction” section for definition of a *warm* start). This restriction is required to prevent FRC errors from occurring because the registers in the master and checker may have different values.

The BXU loads the *FRC-Splitting-Control* register conditionally. A *cold* RESET signal clears the control bits of this register; a *warm*RESET signal does not modify any of the control bits. The *FRC-Splitting-Control* register is only loaded after *cold* RESET signal.

Pins Sampled During Default Initialization

Several pins of the BXU are sampled while RESET is asserted. This sampling occurs continually on the falling edge of the first phase of the internal clock. When the RESET signal is deasserted, the BXU uses the last values sampled for initialization.

The chosen default values allow the BXU to function in a simple system configuration without requiring any external logic devices. The default values of each register are listed in Appendix A. The AP-bus and L-bus parameter settings are described in the following paragraphs.

AP-Bus Parameters

If the $\overline{\text{COM}}$ pin is high during RESET, then the BXU is a bus master in a master/checker FRC pair. The master drives the AP-bus, and the checker ($\overline{\text{COM}}$ low) monitors the master and reports any disagreement between the two 80960MC processors.

L-Bus and Module Parameters

IAC requests on the L-bus use the contents of the *System-Bus-ID* register for the *L-Bus* destination field (see Appendix A for the description of the *System-Bus-ID* register). The *System-Bus-ID* register assigns a unique identification to each BXU in the same module. The contents of this register are also used to identify the AP-bus for the BXU. The *System-Bus-ID* register is read from the $\overline{\text{MODCHK}}$ and $\overline{\text{BOUT}}$ lines during RESET. Table 14-1 shows the identification assignment for different values of $\overline{\text{MODCHK}}$ and $\overline{\text{BOUT}}$.

One $\overline{\text{LERL}}$ line is used to select the mode of operation on the BXU. If the $\overline{\text{LERL}}$ pin is high, then the *BXU-Mode* bit in the *LBI-Control* register is set to a value of one to select Processor mode. Otherwise, the BXU operates in Memory mode.

Table 14-1: System-Bus-Identification Assignments

System-Bus-ID Register		MODCHK	BOUT
Bit ₁₅	Bit ₁₄		
1	1	0	0
1	0	0	1
0	1	1	0
0	0	1	1

Loading Parameters

For parameters different from the default values, an external controller, such as the M8051, can be used to load the appropriate registers. As an alternative, a three-state buffer can be used by setting the inputs to the appropriate high or low values with pull-up or pull-down resistor networks. The three-state buffer can be enabled by RESET making the parameters valid during RESET. The buffer can be disabled after RESET is deasserted.

The parameters must be valid four full bus cycles (eight CLK2 cycles) prior to the trailing edge of RESET. To accommodate relatively slow external controllers, the parameters may remain asserted on the pins until bus traffic begins on either the L-bus or the AP-bus of the BXU. Thus, the module with the fastest RESET logic must not generate AP-bus traffic until the slowest module has released its RESET parameters. In general, this condition is met when the 80960MC processor performs the self-test function during initialization.

The BXU disables the error reporting operation when it receives a RESET signal. This action prevents any reaction based on the parameters loaded on BOUT and MODCHK. Furthermore, the COM and LERL₁-LERL₀ lines are negative edge sensitive signals. Releasing the parameter can only cause a positive going edge. Thus, no reaction occurs in the BXU.

BXU State at the End of RESET

At the end of RESET each BXU in the module has a unique value in the *System-Bus-ID* register and has the IAC recognition function enabled. A 80960MC processor on the L-bus can use IAC type 0000_B (*Request to BXUs on L-bus*) to address the BXU on the message bus. In this way, the 80960MC processor can access more information about the board configuration by using the serial protocol with the COM register. Note that until the board configuration is known, the 80960MC processor can only address the message BXU (the BXU on AP-bus₀), because it is the only BXU guaranteed to be present.

After the board configuration is known, the 80960MC processor can access all the BXUs by IAC register requests using a logical or physical address. The module address in both cases is always zero,

which is the default value for both the *Physical-ID* and *Logical-ID* registers. These IAC requests are handled entirely within the module without generating AP-bus traffic.

The INIT-RAM memory recognizer is enabled because the *Disable-INIT-RAM* bit in the *LBI-Control* register is zero (default value). This recognizer matches any memory request that has \overline{AD}_{31} asserted and \overline{AD}_{30} deasserted. Addresses in this space are mapped into the external SRAM normally used by the cache. Thus, the lower half of memory can be made available for PROM storage and the upper-addresses can be reserved for memory used during initialization. The INIT-RAM memory recognizer can be disabled by setting the *Disable-INIT-RAM* bit in the *LBI-Control* register, if this function is not desired.

Requests that match the INIT-RAM memory recognizer are handled by the local SRAM, and none of these requests flow onto the AP-bus. The BXU sequences the \overline{READY} and $\overline{WD}_1\text{--}\overline{WD}_0$ signals, and maps the LAD_{16} and LAD_{15} lines to the \overline{WY}_0 and \overline{WY}_1 lines, respectively. This provides up to 128K bytes of RAM storage (with four AP-buses) for use by initialization software. The signal sequencing uses the slow-read, slow-write cache timing mode (specified by the *Cache-Configuration* register).

During RESET, the BXU enables the IAC address recognizers for the AP-bus and disables the memory address recognizer. In addition, the BXU disables the cache and prefetch units and marks the entire cache directory invalid. The BXU does not initiate any requests on either bus. It only responds to requests that it receives.

The BXU does not send error reports, but it correctly responds to any error reports that it receives. All BXUs have the correct *Confinement-ID* field in the *Bus-Error-ID* register. The *Confinement-ID* field in their *Module-Error-ID* register contains FF_{16} . The *Error-Reporting-Enable* bit in the *FTI* register and the *Bus-Switch-Disable* bit in the *AP-Control* register are cleared. Before operating on the AP-bus and using the error reporting network, the software should load information into two registers: the *ID-Parity* bit and the *Source-ID* field in the *Bus-Error-ID* register, and a unique *Confinement-ID* field in the *Module-Error-ID* register.

Identification Phase

The identification phase of initialization for the BXU is required in systems containing more than one BXU. During this phase of initialization, a new *Component-ID* field in the *Physical-ID* register is assigned by the Initialization processor (see Chapter 3). This becomes the physical identification for the BXU.

After default initialization is complete, the Identify Device Order IAC (IAC type 0111_B) is used to load AP-bus identification information into the BXU. Sending an Identify Device Order causes a two-word write request to be generated on the AP-bus. If the \overline{INITID} pin of the BXU is asserted during the first data cycle of this command, the BXU accepts the Identify Device Order command. This special command does not cause the BXU to send a reply.

The \overline{INITID} pin of a BXU is wired to one of the AP-bus address/data lines, as shown in Figure 14-2. The particular A/D line used is an arbitrary decision. For BXUs functioning as a master/checker pair, the \overline{INITID} pins are tied together.

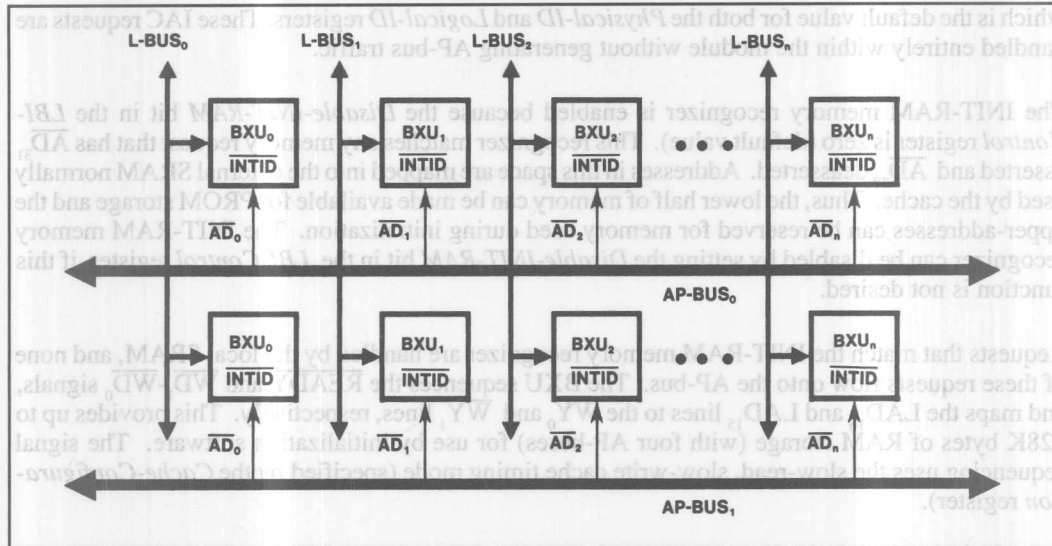


Figure 14-2: How The Identify Device Order Specifies A Particular BXU

An encoded value is loaded from the data pins on the next data cycle to various registers in the BXU. Figure 14-3 shows the fields in the data words associated with this command. The *IIIII* field of the second data word is loaded into the *Component-ID* field of the *Physical-ID* register and the *Unit-ID* field of the *Logical-ID* register. The *DD* field is loaded into the *Drive* field of the *Arbitration-ID* register, and the *CCCC* field is loaded into the *Count* field of the *Arbitration-ID* register (see Appendix A for the complete description of the *Arbitration-ID* register).

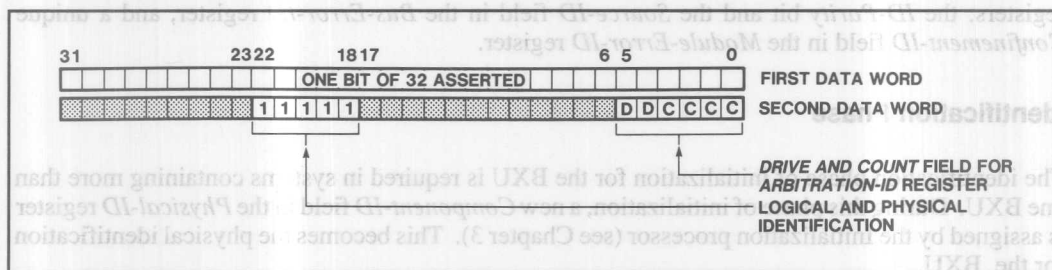


Figure 14-3: Data Field For The Identify Device Order

The use of the Identify Device order is restricted to special situations because of its special address recognition function. A BXU that receives this command must either be idle or processing another Identify Device Order. No other IAC or memory requests may be present in the BXU while it handles the Identify Device Order.

This restriction is necessary because there may be more than one Initialization processor, each responsible for a portion of the total system initialization task. The identification phase of

initialization must be completed by all Initialization processors before they begin the parameterization phase of initialization.

After sending an Identify Device Order, a BXU (or FRC pair of BXUs) has the same *Unit-ID* field in the *Logical-ID* register and *Component-ID* in the *Physical-ID* register. The parameterization phase of initialization can be used to change the *Unit-ID* field of the *Logical-ID* register. These identification fields are used to address the BXUs during normal system operation. The assignment of these identification fields is arbitrary.

The *Arbitration-ID* register is used during normal operation to indicate a unique time when a BXU drives a request on the AP-bus. Although particular assignments can be arbitrary unique values, the most efficient value assignments are packed, e.g., 00_H, 10_H, 20_H, 01_H, 11_H, 21_H, ..., 0F_H, 1F_H, 2F_H. Duplicate values in the *Arbitration-ID* registers cause AP-bus contention and loss of pipeline ordering.

Parameterization Phase

The parameterization phase of initialization is only required if the BXU registers must be set to values other than the default values. During this phase of initialization, the default values that were set up during the first phase may be altered and other register values may be changed.

Parameterization consists of two separate activities. During the first activity, external information is sent to the BXU. This occurs during the period in which the COM pin is defined to function as a serial input to the COM register (see Appendix A for the complete description of the COM register). The COM register can be loaded by an external controller with specific board parameters. (The contents of the COM register are strictly arbitrary and it is not necessary to use this register for the parameterization phase if software uses other techniques to change the contents of the BXU registers.)

For example, the controller on a memory board can load the COM register with a value that indicates that this board is a memory board with a certain amount of memory, type, etc. On the other hand, a controller on a processor board could load the COM register of its BXUs with a code that specifies that this board is a processor board, perhaps with a cache, etc. The protocol for loading this register is explained later in this section.

During the second activity in the parameterization phase, the default values are changed consistent to the parameters that were loaded into the COM register. The BXU itself does not use the contents of the COM register to change its default values. Software must read the COM register and determine how to change the registers of the BXU with the new parameters.

Serial COM Protocol

The protocol begins with a start transition, and loads up to 32 bits of information into the COM register. The serial loading of this register depends on a multiple-sample protocol where the sample points are defined by the BXU. The COM protocol for loading a single bit of information from the COM pin is illustrated in Figure 14-4.

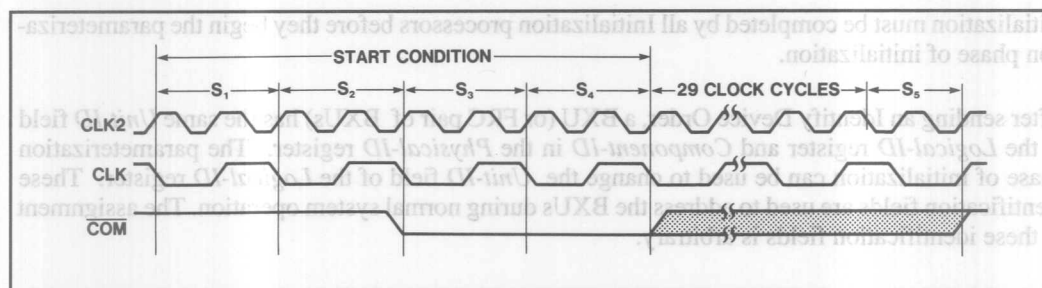


Figure 14-4: COM Pin Protocol

The sequence of events required to load the *COM* register is listed below.

1. The start condition for a bit is detected by the high-to-low transition of the \overline{COM} signal. The BXU detects the start condition when the \overline{COM} pin is high for at least two clock cycles (S_1 and S_2), and then the \overline{COM} is low for at least two clock cycles later (S_3 and S_4). Neither a high nor a low level signal triggers the COM protocol.
2. When the start condition is detected, the following occurs:
 - a. The *Test-Enable* bit in the *Test-Detection* register is cleared.
 - b. The *Com-Register-Altered* bit in the *AP-Control* register is set.
 - c. An *COM-Altered* error report is sent by the BXU on the \overline{BERL}_1 - \overline{BERL}_0 lines if the *Com-Reporting* bit is set in the *FT1* register. This occurs only if the *Com-Register-Altered* bit in the *AP-Control* register was not set prior to the start condition detection.
 - d. Other usages of the *COM* register are disabled. The usages of the \overline{COM} pin and *COM* register include the following:
 - i. Testing the BXU's FRC logic (see Chapter 12).
 - ii. Signalling permanent errors: the BXU drives the \overline{COM} pin when the *Drive-Com* bit in the *AP-Control* register is set, or when the *Faulty* bit in the *FT2* register is set.
 - e. On the twenty-eighth clock cycle following S_4 , the value on the \overline{COM} pin is latched. Note that the \overline{COM} pin may go high at any time following S_5 , and the \overline{COM} pin may go low at any time following S_2 for a new start.
3. For each subsequent start condition, the sampled value is shifted into the *COM* shift chain.

The latched value is shifted into the \overline{COM} shift chain, as illustrated in Figure 14-5. Note that the bits are shifted in a special sequence. The first bit shifted in appears in bit₁₆ of the *COM* register. The last bit shifted in appears in bit₁₅. A total of 38 bits of data must be shifted in, 32 bits are visible and six bits are hidden. This means that software reads 32 bits (bit₃₁ through bit₀) from the *COM* register; the hidden bits are not read.

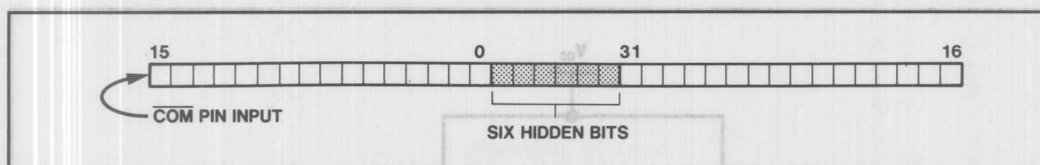


Figure 14-5: *COM* Register Loading By Using The \overline{COM} Pin

Write operations to this *COM* register clear the hidden bits. The hidden bits are part of a shift chain used for testing the FRC logic of the BXU. See Chapter 12, the “FRC Logic” section, for more information on testing the FRC logic.

PROGRAMMING NOTE

A potential race condition exists between software and hardware usage of the *COM* register. The software must wait for the serial data transfer to complete before it reads data from the *COM* register. One way to detect the completion of loading 38 bits is to compare the contents of the *COM* register with the previous contents of the *COM* register, and not clear the *Com-Register-Altered* bit. When there is no change in the data, then the transfer is complete. Another method would be to employ a microcontroller that sets a hardware flag when the transfer is completed.

Every time the *COM-Register-Altered* bit is set, a *COM-Altered* error report is generated by the BXU if the *COM-Reporting* bit in the *FT1* register is set. The *COM* protocol makes no provision for detecting the end of the loading sequence.

Cold and Warm Reset Distinction

The BXU is capable of distinguishing between a power-up RESET (*cold* RESET) and a RESET that comes during normal operation (*warm* RESET). The BXU uses the distinction between a *warm* and *cold* RESET to control the FRC splitting logic. FRC splitting is an option that allows an FRC pair to be split under software control. An FRC split occurs only on *warm* RESET.

The V_{REF} pin of the BXU is used to distinguish between a *warm* or *cold* RESET. To detect a *cold* RESET, an external resistor/capacitor circuit on the V_{REF} pin can be designed to pull the signal above $V_{CC} - 0.7$ Volts (called the cold trip point) at the time of power-up, as shown in Figure 14-6. The values of the resistor/capacitor network must be chosen such that the V_{REF} pin is held above the *cold* trip point until the RESET signal reaches a stable voltage level (either high or low). After RESET reaches a stable voltage, the V_{REF} pin is brought between 1.7 Volts and 3.3 Volts (called the trip point). RESET must be stable during the transition of V_{REF} from the *cold* trip point to the trip point, as shown in Figure 14-7. The first high-to-low transition of RESET after V_{REF} drops below the trip point is considered a *cold* start. Assertions of RESET after the first high-to-low transition are considered *warm* RESETs. All outputs of the BXU float to a high impedance state on the leading edge of RESET. RESET can be asserted at any time.

If V_{REF} is brought above the maximum trip point value (3.3 Volts) again, it must be followed by a RESET sequence. The first assertion of RESET after the V_{REF} pin has reached the trip point is labeled a *cold* RESET. All subsequent RESET signals are considered *warm* RESETs. After the leading edge of every RESET pulse, the *Warm-RESET* bit in the *FRC-Splitting-Control* register indicates whether the initialization was *warm* or *cold* RESET.

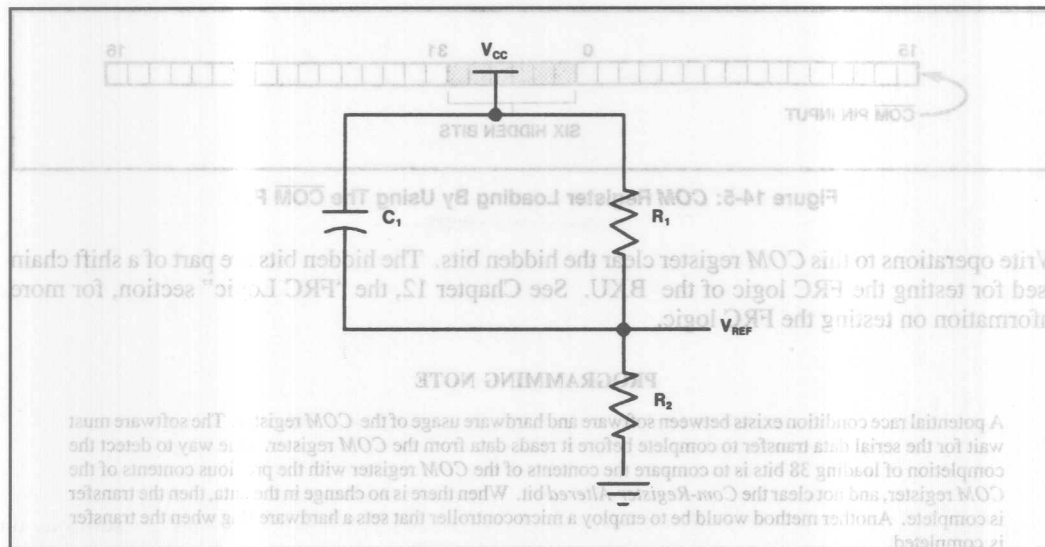


Figure 14-6: RC Network For V_{REF}

Every time the COM-Register-Address is generated by the BXU it is the COM-Register bit in the FTY register is set. The COM protocol makes no provision for detecting the end of the loading sequence.

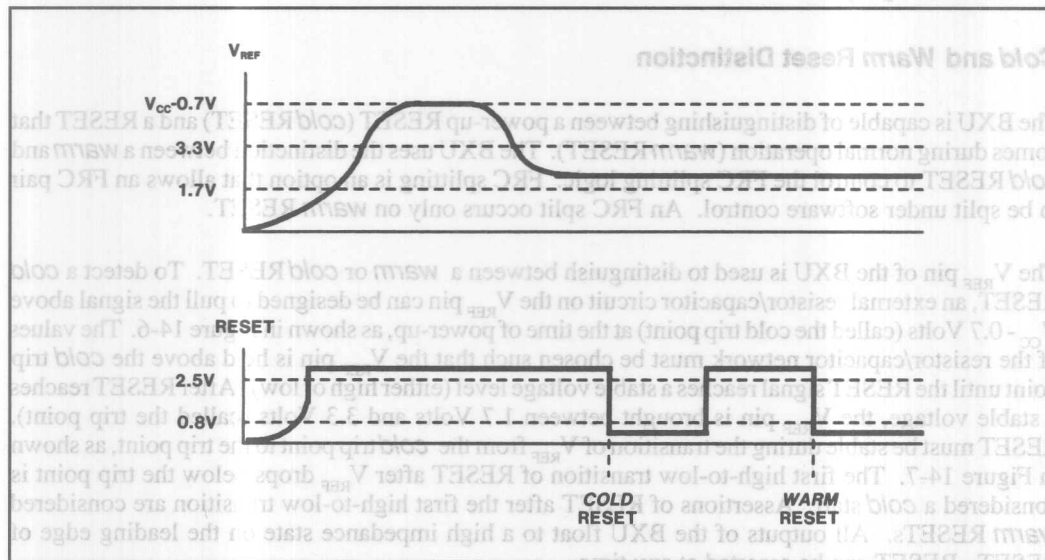


Figure 14-7: Relationship Between V_{REF} And RESET

The V_{REF} pin of the BXU is used to distinguish between a warm or cold RESET. To detect a cold RESET, an external resistor/capacitor circuit on the V_{REF} pin can be designed to pull the signal above $V_{CC} - 0.7$ Volts (called the cold trip point) at the time of power-up, as shown in Figure 14-7. The values of the resistor/capacitor network must be chosen such that the V_{REF} pin is pulled above the cold trip point until the RESET signal reaches a stable voltage level (either high or low). After RESET reaches a stable voltage level, the V_{REF} pin is brought back to 1.7 Volts and 3.3 Volts (called the trip point). RESET must be stable during the transition from the cold trip point to the warm trip point, as shown in Figure 14-7. The first high-to-low transition of RESET after V_{REF} drops below the trip point is considered a cold RESET. All subsequent RESET signals are considered warm RESET. All outputs of the BXU float to a high impedance state on the leading edge of every RESET pulse, the Warm-Reset bit in the FRC-Splitting-Control register indicates whether the initialization was warm or cold RESET.

PROGRAMMING NOTE

V_{REF} must be brought to $V_{REF} - 0.7$ Volts in order to reset the *FRC-Splitting-Control* register, otherwise it will be loaded with random values.

MODULE SHADOWING

Primary/shadow pairs can be married during initialization or during system operation using another module or a special agent. Chapter 13 describes how to marry these pairs during system operation. The following paragraphs show how to marry a primary/shadow pair during initialization using a special agent.

Marrying a Primary/Shadow Pair Using a Special Agent

Figure 14-8 shows the configuration that marries a primary/shadow pair during initialization using a special agent. During RESET, the 80960MC processor in *Primary-Elect* unit is setup as the initialization processor (a high voltage on the STARTUP pin, see Chapter 3), while the 80960MC processor in the *Shadow-Elect* is setup as the non-initialization processor. Thus, the *Primary-Elect* performs all of the necessary register configuration for itself and the 80960MC processor in the *Shadow-Elect* unit.

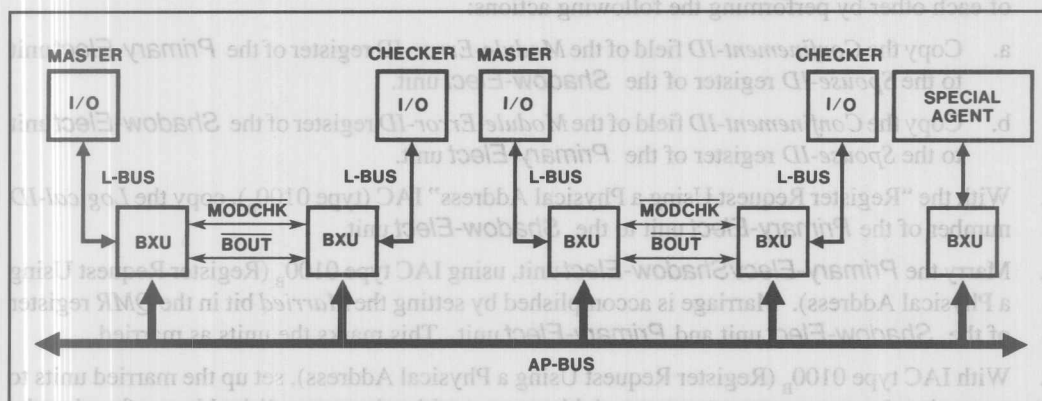


Figure 14-8: Marrying A Primary/Shadow Pair Using A Special Agent

After completing the register configuration, the processor in the *Primary-Elect* signals the special agent to send a "Restart Processor" message IAC using the logical address of the primary/shadow pair (see the *80960MC Programmer's Reference Manual* for more details on the different types of message IAC requests). This special agent delays sending the message IAC to allow enough time for the processor in the *Primary-Elect* unit to send a "Stop-Processor" message IAC to itself causing it to enter the stopped state.

The following paragraphs enumerate the steps that the initialization processor performs to configure the primary/shadow pair for marriage. These steps follow the normal identification phase of the BXU

initialization. If there are multiple buses in the system, any action performed on one BXU in a module is performed on all BXUs in that module.

1. With the "Register Request Using a Physical Address" IAC (type 0100_B), set the *Shadow* bit in the *QMR* register of the shadow BXU.
2. Copy the *AP-Mask* and *AP-Match* registers from the BXU in the *Primary-Elect* unit to the BXU in the *Shadow-Elect* unit.
3. With the "Register Request Using a Physical Address" IAC (type 0100_B), prepare the shadow for marriage by performing the following actions:

- a. Copy the contents of the registers in the *Primary-Elect* unit to the registers of the *Shadow-Elect* unit except for the *QMR*, *Spouse-ID*, *Module-Error-ID*, *Bus-Error-ID*, *Logical-ID*, and *Physical-ID* registers.

Because the *Arbitration-ID* register is write-only register, software must write to both registers with the same value.

- b. Set the *Module-Error-ID* register in the *Shadow-Elect* unit to a unique value.
 - c. Set up the *Bus-Error-ID* register in the *Shadow-Elect* unit by setting the *Source-ID* to a unique value.
4. With the "Register Request Using a Physical Address" IAC (type 0100_B), make the units aware of each other by performing the following actions:

- a. Copy the *Confinement-ID* field of the *Module-Error-ID* register of the *Primary-Elect* unit to the *Spouse-ID* register of the *Shadow-Elect* unit.
- b. Copy the *Confinement-ID* field of the *Module-Error-ID* register of the *Shadow-Elect* unit to the *Spouse-ID* register of the *Primary-Elect* unit.

5. With the "Register Request Using a Physical Address" IAC (type 0100_B), copy the *Logical-ID* number of the *Primary-Elect* unit to the *Shadow-Elect* unit.
6. Marry the *Primary-Elect/Shadow-Elect* unit, using IAC type 0100_B (Register Request Using a Physical Address). Marriage is accomplished by setting the *Married* bit in the *QMR* register of the *Shadow-Elect* unit and *Primary-Elect* unit. This marks the units as married.
7. With IAC type 0100_B (Register Request Using a Physical Address), set up the married units to recognize the same memory ranges. Address recognition is accomplished by performing the following functions:
 - a. Set the *AP-Match* registers to the desired value.
 - b. Set the *AP-Mask* registers to the desired value.
 - c. Set the *Enable* bit in the *AP-Match* registers.

Special Agent Overview

After completing the register configuration, the processor in the *Primary-Elect* signals the special agent to send a "Restart Processor" message IAC using the logical address of the primary/shadow

pair. This special agent delays sending the message IAC to allow enough time for the processor in the *Primary-Elect* unit to send a "Stop-Processor" message IAC to itself causing it to enter the stopped state.

Figure 14-9 illustrates one way in which to implement this special agent. The logic sends a “Restart-Processor” IAC message after a delay of approximately 20 μ s. The logic can be attached to the L-bus of a any BXU in Processor mode (the BXU has to be in Processor mode in order to drive the AP-bus).

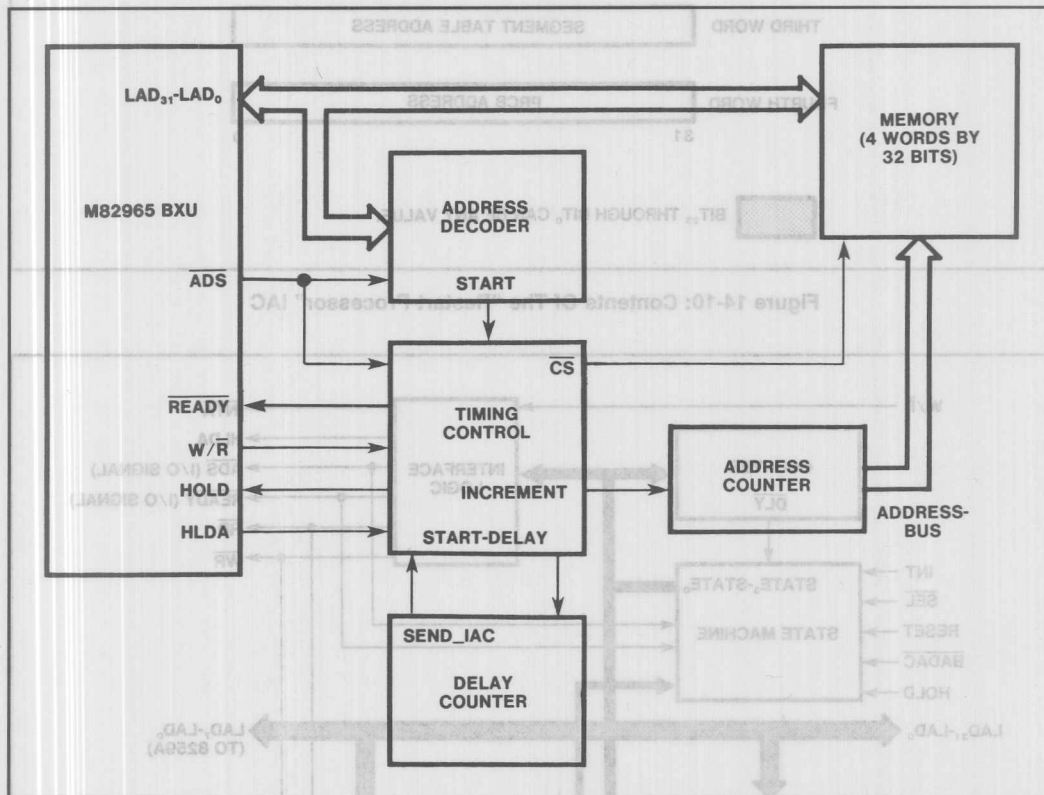


Figure 14-9: Special Agent Logic

Figure 14-10 shows the contents IAC message that is stored in memory. The first word access contains the IAC message address format (this is the same format as described in Chapter 7). The second word access identifies the “Restart-Processor” IAC message. The third and fourth word accesses contain the segment table address and Processor Control Board (PRCB) address, respectively (see the *80960MC Programmer’s Reference Manual* for more information on the this type of IAC message).

The IAC generator described in Chapter 9 can be used for the special agent by making two modifications: the addition of delay times to allow the processor enough time to send a “Stop-

Processor" IAC to itself and the capability to generate a "Restart Processor" IAC. The configuration is shown again in Figure 14-11.

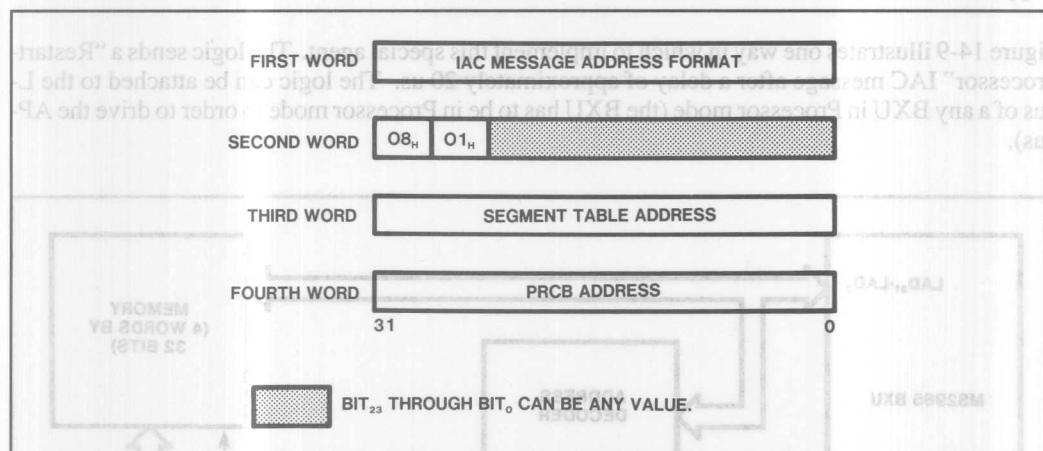


Figure 14-10: Contents Of The "Restart-Processor" IAC

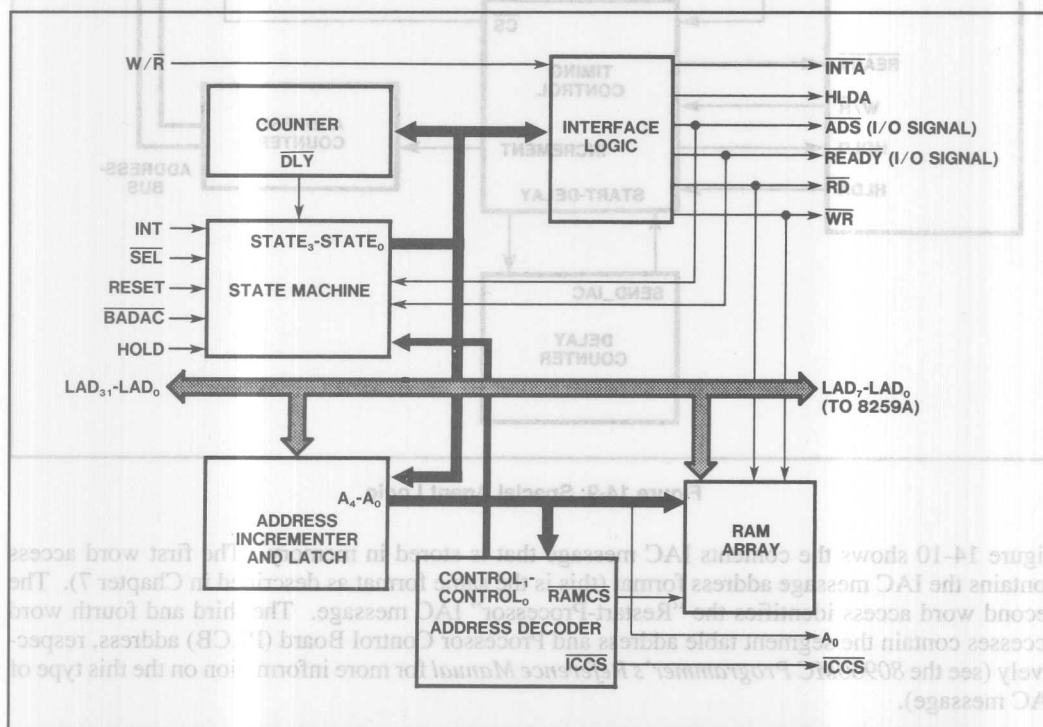


Figure 14-11: IAC Generation Logic

Table 14-2: Modified State Functions

State	Name	Signals Asserted	Other Action	Comments
I	Idle	None	Set MASK bit after reset	Waiting for INT from M8259A, or HOLD from BXU
A ₀	Address ₀	HLDA	Latch LAD ₄ -LAD ₀ if ADS is asserted	Grant bus to BXU; place ADS in high impedance
A ₁	Address ₁	HLDA	Decode latched address	Wait here if SEL not asserted
DA ₀	Data Access ₀	HLDA, RD, or WR	Start delay timer	Wait for DLY to meet M8259A TRLRH spec
DA ₁	Data Access ₁	HLDA, RD, or WR, READY	Set or reset MASK bit if commanded	Latch data at end of cycle if write operation; assert READY to complete transfer
BW	Bus Wait	HLDA, READY	None	Wait for BXU to release L-Bus
CW	Counter Wait	None	None	Wait for 400 clock cycles to time-out
IM ₀	IAC Message Address	ADS, RD	Increment IAC RAM address at end of cycle	Send IAC address; place READY in high impedance
IM ₁	IAC Message Data Word ₁	RD	Increment IAC RAM address at end of cycle	Send Data Word ₁ ; increment address Modulo 4 if no restart, otherwise Modulo 4
IM ₂	IAC Message Data Word ₂	RD	Increment IAC RAM address at end of cycle	Send Data Word ₂ ; increment address Modulo 4
IM ₃	IAC Message Data Word ₃	RD	Increment IAC RAM address at end of cycle	Send Data Word ₃ ; increment address Modulo 4
CK	Check for BADAC	None	None	If BADAC asserted retry IAC message
IA	Interrupt Acknowledge	INTA	None	Wait for DLY to meet M8259A TRLRH spec
II	Interrupt Idle Time	None	None	Wait for DLY to meet M8259A TRHRL spec

NOTE:

* Since each interrupt IAC message occupies two words in RAM, the vector must be shifted left by 1 bit in order to serve as a valid IAC message address.

Table 14-2: Modified State Functions (cont.)

State	Name	Signals Asserted	Other Action	Comments
VF ₀	Interrupt Vector Fetch ₀	INTA	*Latch shifted vector at end of cycle	Wait for DLY to meet M8259A TRLRH spec
VF ₁	Interrupt Vector Fetch ₁	INTA	None	Interrupt vector decode wait

NOTE:

* Since each interrupt IAC message occupies two words in RAM, the vector must be shifted left by 1 bit in order to serve as a valid IAC message address.

QMR INITIALIZATION EXAMPLE

This section illustrates the initialization process with a step-by-step example. The system configuration consists of a processor primary/shadow pair and an I/O primary/shadow pair, as shown in Figure 14-13. In this example, there is no cache and no global memory. The I/O module contains memory that is accessed periodically by monitor sensors. The I/O module also has logic to generate a "Restart Processor" message IAC, which is triggered by a write operation to a particular location in the I/O space. There are two buses in the system where AP-bus₁ is a passive backup (i.e., no interleaving is done).

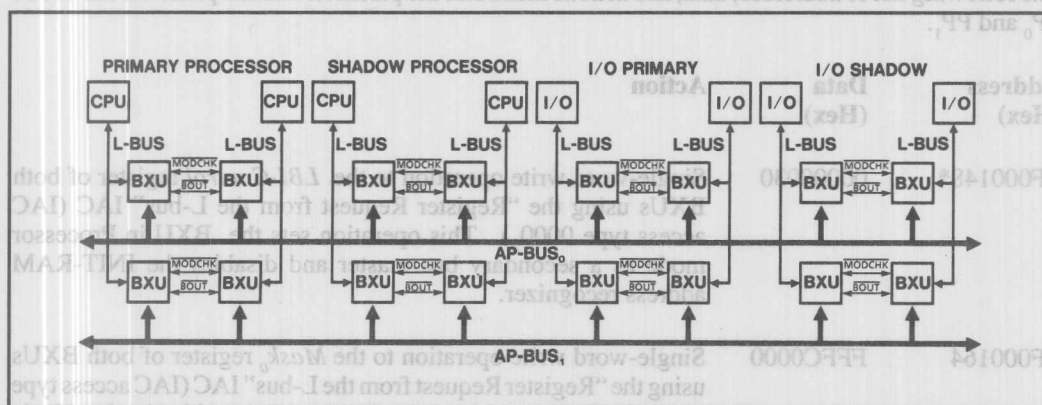


Figure 14-13: Configuration For Example

The names for the eight BXU pairs in the example configuration are shown in the following list:

- PP₀ identifies the BXU pair in the processor *Primary-Elect* unit on AP-bus₀
- PP₁ identifies the BXU pair in the processor *Primary-Elect* unit on AP-bus₁
- PS₀ identifies the BXU pair in the processor *Shadow-Elect* unit on AP-bus₀

- PS_1 identifies the BXU pair in the processor *Shadow-Elect* unit on AP-bus₁
- I/OP_0 identifies the BXU pair in the I/O *Primary-Elect* unit on AP-bus₀
- I/OP_1 identifies the BXU pair in the I/O *Primary-Elect* unit on AP-bus₁
- I/OS_0 identifies the BXU pair in the I/O *Shadow-Elect* unit on AP-bus₀
- I/OS_1 identifies the BXU pair in the I/O *Shadow-Elect* unit on AP-bus₁

The following sections show the appropriate address and data words used to initialize each BXU in this example system configuration. The description first examines the parameterization and identification phases, then describes the marriage sequence. This example follows the step-by-step guidelines presented in the "Module Shadowing" section for the marriage sequence. The 80960MC processor in the *Primary-Elect* unit performs the initialization actions.

Local BXU Parameterization

For this configuration the parameterization phase of initialization is performed on the BXUs in the processor *Primary-Elect* unit prior to the identification phase to establish the error reporting network. This is because the 80960MC processor in the *Primary-Elect* unit serves as the initialization processor for the system. All registers that need to be changed from their RESET default values are initialized except for the *QMR* register. The *QMR* register is initialized as part of the marriage sequence.

The following list of addresses, data, and actions illustrates the parameterization phase for the BXUs PP_0 and PP_1 .

Address (Hex)	Data (Hex)	Action
FF000148*	00000030	Single-word write operation to the <i>LBI-Control</i> register of both BXUs using the "Register Request from the L-bus" IAC (IAC access type 0000 _B). This operation sets the BXU in Processor mode as a secondary bus master and disables the INIT-RAM address recognizer.
FF000164	FFFC0000	Single-word write operation to the <i>Mask₀</i> register of both BXUs using the "Register Request from the L-bus" IAC (IAC access type 0000 _B). This operation sets the appropriate bits in the <i>Mask₀</i> register.
FF000160	00040001	Single-word write operation to the <i>Match₀</i> register of both BXUs using the "Register Request from the L-bus" IAC (IAC access type 0000 _B). This operation enables the bus recovery of the <i>Mask₀</i> and <i>Match₀</i> registers.

Address (Hex)	Data (Hex)	Action
FF000240	00000004	Single-word write operation to the <i>Prefetch-Control</i> register of both BXUs using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation makes both prefetch channels available.
FF000054	00000007	Single-word write operation to the <i>FTI</i> register of both BXUs using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation selects the Wait-for-BERL timing, enables the error reporting network, and ensures proper fault-tolerant operation.
FF0010E4	00000401	Single-word write operation to the <i>Module-Error-ID</i> register of the BXU on AP-bus ₀ using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation sets the <i>Confinement-ID</i> field to a value of four and the <i>Source-ID</i> field to a value of zero.
FF0050E4	00000402	Single-word write operation to the <i>Module-Error-ID</i> register of the BXU on AP-bus ₁ using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation sets the <i>Confinement-ID</i> field to a value of four the <i>Source-ID</i> field to a value of one.
FF0008E8	00000009	Single-word write operation to the <i>Bus-Error-ID</i> register of the BXU on AP-bus ₀ using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation sets the <i>Confinement-ID</i> field to a value of zero the <i>Source-ID</i> field to a value of four.
FF0048E8	00000108	Single-word write operation to the <i>Bus-Error-ID</i> register of the BXU on AP-bus ₁ using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation sets the <i>Confinement-ID</i> field to a value of one the <i>Source-ID</i> field to a value of four.
FF0000C4	00000500	Single-word write operation to the <i>Spouse-ID</i> register of both BXUs using the “Register Request from the L-bus” IAC (IAC access type 0000 _B). This operation sets the <i>Spouse-ID</i> field to a value of five.

*Note: The following sample of assembly code can be used for the first write operation. All other write operations can use the same sequence of instructions.

Instruction	Address/Data	Comment
LDCONST	0xFF000148,G ₀	Load the address in register G ₀
LDCONST	0x00000030,G ₁	Load the data in register G ₁
SYNMOV	G ₀ ,G ₁	Move data to the address contained in register G ₀
BE	ERROR	Report an error if it occurs

Identification Phase and Primary/Shadow Marriage

The identification phase is divided into two sections: the assignment of identification values for each BXU and the initialization of the I/O primary/shadow units. These topics and the marriage of the processor primary/shadow units are described in this section.

BXU Identification Assignment

During this phase, six BXU pairs are assigned logical, physical, and arbitration identification values by using the "Identify Device Order" IAC. The primary processor BXUs on AP-bus₀ and AP-bus₁ use the default values of the appropriate registers. Consequently these BXUs are not assigned new values.

Address (Hex)	Data (Hex)	Action
FF001C00	00000002 00040000	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the PS ₀ BXU using the "Identify Device Order" IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 01 _H and the <i>Drive</i> and <i>Count</i> fields of the <i>Arbitration-ID</i> register are set to a value of zero. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 01 _H , but this value is changed later.
FF001C00	00000004 00080001	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the I/OP ₀ BXU using the "Identify Device Order" IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 02 _H , the <i>Drive</i> field of the <i>Arbitration-ID</i> register to a value of zero, and the <i>Count</i> field of the <i>Arbitration-ID</i> register to a value of one. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 02 _H , but this value is changed later.

Address (Hex)	Data (Hex)	Action
FF001C00	00000008 000C0001	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the I/O ₀ BXU using the “Identify Device Order” IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 03 _H , the <i>Drive</i> field of the <i>Arbitration-ID</i> register to a value of zero, and the <i>Count</i> field of the <i>Arbitration-ID</i> register to a value of one. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 03 _H , but this value is changed later.
FF005C00	00000002 00040000	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the PS ₁ BXU using the “Identify Device Order” IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 01 _H , the <i>Drive</i> field of the <i>Arbitration-ID</i> register to a value of zero, and the <i>Count</i> field of the <i>Arbitration-ID</i> register to a value of zero. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 01 _H , but this value is changed later.
FF005C00	00000004 00080001	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the I/O _P ₁ BXU using the “Identify Device Order” IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 02 _H , the <i>Drive</i> field of the <i>Arbitration-ID</i> register to a value of zero, and the <i>Count</i> field of the <i>Arbitration-ID</i> register to a value of one. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 02 _H , but this value is changed later.
FF005C00	00000008 000C0001	Two-word write operation to the <i>Logical-ID</i> , <i>Physical-ID</i> , and <i>Arbitration-ID</i> registers of the I/O _S ₁ BXU using the “Identify Device Order” IAC (IAC access type 0111 _B). The <i>Component-ID</i> field of the <i>Physical-ID</i> register is set to a value of 03 _H , the <i>Drive</i> field of the <i>Arbitration-ID</i> register to a value of zero, and the <i>Count</i> field of the <i>Arbitration-ID</i> register to a value of one. The <i>Unit-ID</i> field of the <i>Logical-ID</i> register is also set to a value of 03 _H , but this value is changed later.
FF041044	00000000	Single-word write operation to the <i>Logical-ID</i> register of the PS ₀ BXU employing the “Register Request Using a Physical Address” IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> to a value of zero.
FF045044	00000000	Single-word write operation to the <i>Logical-ID</i> register of the PS ₁ BXU employing the “Register Request Using a Physical Address” IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> field to a value of zero.

Initializing the BXUs in the I/O Primary/Shadow Units

For the I/O primary and shadow units, all the registers of the BXUs that require change from their RESET default values are initialized at this time including the *QMR* registers. The I/O modules are married with the toggling enabled. This section presents the initialization steps for each BXU in the I/O Primary-Elect/Shadow-Elect units.

Initializing the I/O Primary-Elect BXU on AP-Bus₀

Address (Hex)	Data (Hex)	Action
FF081044	00040000	Single-word write operation to the <i>Logical-ID</i> register of the I/OP ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> field to a value of one.
FF0410E4	00000600	Single-word write operation to the <i>Module-Error-ID</i> register of the I/OP ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of six and <i>Source-ID</i> field to a value of zero.
FF0810E8	0000000C	Single-word write operation to the <i>Bus-Error-ID</i> register of the I/OP ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of zero and the <i>Source-ID</i> field to a value of six.
FF0810C4	00000700	Single-word write operation to the <i>Spouse-ID</i> register of the I/OP ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of seven.
FF0810E0	0000000E	Single-word write operation to the <i>QMR</i> register of the I/OP ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets <i>Married</i> , and <i>Toggle-Primary/Shadow</i> control bits. Note that the write enable for the <i>Married</i> bit is turned on.

Initializing the I/O Shadow-Elect BXU on AP-Bus₀

Address (Hex)	Data (Hex)	Action
FF0C1044	00040000	Single-word write operation to the <i>Logical-ID</i> register of the I/O ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> field to a value of one.
FF0C10E4	00000701	Single-word write operation to the <i>Module-Error-ID</i> register of the I/O ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of seven and the <i>Source-ID</i> field to a value of zero.
FF0C10E8	0000000F	Single-word write operation to the <i>Bus-Error-ID</i> register of the I/O ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of zero and the <i>Source-ID</i> field to a value of seven.
FF0C10C4	00000600	Single-word write operation to the <i>Spouse-ID</i> register of the I/O ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of six.
FF0C10E0	0000003E	Single-word write operation to the <i>QMR</i> register of the I/O ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Married</i> , <i>Toggle-Primary/Shadow</i> , and <i>Shadow</i> control bits. Note that the write enable for the <i>Shadow</i> and <i>Married</i> bits is turned on.

Initializing the I/O *Primary-Elect* and *Shadow-Elect* BXUs on AP-Bus₀)

The “Register Request Using a Logical Address” IAC can be used to write to both the I/O *Primary-Elect* and *Shadow-Elect* BXUs. Although this action saves some programming steps, it is dangerous to use “Register Request Using a Logical Address” IAC on the AP-bus before the *Unit-ID* field in the *Logical-ID* register is assigned in all BXUs.

Address (Hex)	Data (Hex)	Action
FF04086C	FFFC0000	Single-word write operation to the <i>AP-Mask</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation sets the address recognition range and configures the BXU for no interleaving.
FF040868	00040001	Single-word write operation to the <i>AP-Match</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation partitions the address recognition range and configures the BXU for no interleaving.
FF040948	000000F0	Single-word write operation to the <i>LBI-Control</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation sets the BXU that is in Processor mode as a primary bus master and disables the INIT-RAM memory recognizers.
FF040854	00000007	Single-word write operation to the <i>FTI</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation selects the Wait-for-BERL timing, enables the error reporting network, and ensures proper fault-tolerant operation.

Initializing the I/O Primary-Elect BXU on AP-Bus

Address (Hex)	Data (Hex)	Action
FF085044	00040000	Single-word write operation to the <i>Logical-ID</i> register of the I/OP ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> field to a value of one.
FF0850E4	00000603	Single-word write operation to the <i>Module-Error-ID</i> register of the I/OP ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of six and the <i>Source-ID</i> field to a value of one.
FF0850E8	0000010D	Single-word write operation to the <i>Bus-Error-ID</i> register of the I/OP ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of one and the <i>Source-ID</i> field to a value of six.
FF0850C4	00000700	Single-word write operation to the <i>Spouse-ID</i> register of the I/OP ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of seven.
FF0850E0	0000000E	Single-word write operation to the <i>QMR</i> register of the I/OP ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Married</i> , and <i>Toggle-Primary/Shadow</i> control bits. Note that the write enable for the <i>Married</i> bit is turned on.

Initializing the I/O Shadow-Elect BXU on AP-Bus₁

Address (Hex)	Data (Hex)	Action
FF0C5044	00040000	Single-word write operation to the <i>Logical-ID</i> register of the I/OS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Unit-ID</i> field to a value of one.
FF0C50E4	00000702	Single-word write operation to the <i>Module-Error-ID</i> register of the I/OS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of seven and the <i>Source-ID</i> field to a value of one.
FF0C50E8	0000010E	Single-word write operation to the <i>Bus-Error-ID</i> register of the I/OS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of one and the <i>Source-ID</i> field to a value of seven.
FF0C50C4	00000600	Single-word write operation to the <i>Spouse-ID</i> register of the I/OS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of six.
FF0C50E0	0000003E	Single-word write operation to the <i>QMR</i> register of the I/OS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Married</i> , <i>Toggle-Primary/Shadow</i> , and <i>Shadow</i> control bits. Note that the write enable for the <i>Shadow</i> and <i>Married</i> bits is turned on.

Initializing the I/O *Primary-Elect* and *Shadow-Elect* BXUs on AP-Bus

The “Register Request Using a Logical Address” IAC can be used to write to both the I/O *Primary-Elect* and *Shadow-Elect* BXUs on AP-bus. Although this action saves some programming steps, it is dangerous to use “Register Request Using a Logical Address” IAC on the AP-bus before the *Unit-ID* field in the *Logical-ID* register is assigned in all BXUs.

Address (Hex)	Data (Hex)	Action
FF04486C	FFFC0000	Single-word write operation to the <i>AP-Mask</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation sets the address recognition range and configures the BXU for no interleaving.
FF044868	00040001	Single-word write operation to the <i>AP-Match</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation partitions the address recognition range and configures the BXU for no interleaving.
FF044948	000000F0	Single-word write operation to the <i>LBI-Control</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation sets the BXU that is in Processor mode as a primary bus master, and disables the INIT-RAM memory recognizers.
FF044854	00000007	Single-word write operation to the <i>FTI</i> register of the both BXUs employing the “Register Request Using a Logical Address” IAC (IAC access type 0010 _B). This operation selects the Wait-for-BERL timing, enables the error reporting network, and ensures proper fault-tolerant operation.

Processor *Primary-Elect* and *Shadow-Elect* Marriage

The BXUs in the I/O *Primary* and *Shadow* units are already married and awaiting commands from the 80960MC processor. The BXUs in the processor *Primary* and *Shadow* units are initialized except for the *QMR* register. The steps outlined in the “Module Shadowing” section of this chapter are followed in this example to complete the marriage of the processor *Primary* and *Shadow* units. Each number in the “Step” column corresponds to the appropriate step outlined in the “Module Shadowing” section.

After the marriage is completed, the 80960MC processor in the *Primary* unit sends a command to the special agent, which in turn, sends a “Restart Processor” IAC to the 80960MC processor in the *Primary/Shadow* modules. The pair enables toggling by performing a write operation to their respective registers using a “Register Request From the L-bus” IAC.

Step	Address (Hex)	Data (Hex)	Action
1.	FF0410E0	00000030	Single-word write operation to the <i>QMR</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation clears <i>Toggle-Primary/Shadow control</i> bit and sets the <i>Shadow</i> bit. Note that write enable for the <i>Shadow</i> bit is turned on.
	FF0450E0	00000030	Single-word write operation to the <i>QMR</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation clears the <i>Toggle-Primary/Shadow control</i> bit and sets the <i>Shadow</i> bit. Note that the write enable for the <i>Shadow</i> bit is turned on.
2.			No action required because the system configuration does not have global memory.
3A.	FF041148	00000030	Single-word write operation to the <i>LBI-Control</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the BXU that is in Processor mode as a secondary bus master and disables the INIT-RAM address recognizer.
	FF045148	00000030	Single-word write operation to the <i>LBI-Control</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the BXU that is in Processor mode as a secondary bus master and disables the INIT-RAM address recognizer.
	FF041164	FFFC0000	Single-word write operation to the <i>Mask₀</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the appropriate bits in the <i>Mask₀</i> register.
	FF045164	FFFC0000	Single-word write operation to the <i>Mask₀</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the appropriate bits in the <i>Mask₀</i> register.

Step	Address (Hex)	Data (Hex)	Action
	FF041160	00040001	Single-word write operation to the <i>Match₀</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation enables the bus recovery of the <i>Mask₀</i> and <i>Match₀</i> registers.
	FF045160	00040001	Single-word write operation to the <i>Match₀</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation enables the bus recovery of the <i>Mask₀</i> and <i>Match₀</i> registers.
	FF041240	00000004	Single-word write operation to the <i>Prefetch-Control</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation makes both prefetch channels available.
	FF045240	00000004	Single-word write operation to the <i>Prefetch-Control</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation makes both prefetch channels available.
	FF041054	00000007	Single-word write operation to the <i>FTI</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation selects the Wait-for-BERL timing, enables the error reporting network, and ensures proper fault-tolerant operation.
	FF045054	00000007	Single-word write operation to the <i>FTI</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation selects the Wait-for-BERL timing, enables the error reporting network, and ensures proper fault-tolerant operation.
3B.	FF0410E4	00000500	Single-word write operation to the <i>Module-Error-ID</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of five and the <i>Source-ID</i> field to a value of zero.
	FF0450E4	00000503	Single-word write operation to the <i>Module-Error-ID</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of five and the <i>Source-ID</i> field to a value of one.

Step	Address (Hex)	Data (Hex)	Action	Data (Hex)	Address (Hex)	Step
3C.	FF0410E8	0000000A	Single-word write operation to the <i>Bus-Error-ID</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of zero and the <i>Source-ID</i> field to a value of five.			
	FF0450E8	0000010B	Single-word write operation to the <i>Bus-Error-ID</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Confinement-ID</i> field to a value of one and the <i>Source-ID</i> field to a value of five.			
4A.	FF0410C4	00000400	Single-word write operation to the <i>Spouse-ID</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of four.			
	FF0450C4	00000400	Single-word write operation to the <i>Spouse-ID</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Spouse-ID</i> field to a value of four.			
4B.			This step was done in the parameterization phase.			
5.			This step was done in the identification phase.			
6.	FF0000E0	0000000E	Single-word write operation to the <i>QMR</i> register of the PP ₀ and PP ₁ BXUs using the "Register Request From the L-bus" IAC (IAC access type 0000 _B). This operation sets the <i>Married</i> , and <i>Toggle-Primary/Shadow</i> control bits. Note that the write enable for the <i>Married</i> bit is turned on.			
	FF0410E0	0000003E	Single-word write operation to the <i>QMR</i> register of the PS ₀ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Married</i> , <i>Toggle-Primary/Shadow</i> , and <i>Shadow</i> control bits. Note that the write enable for the <i>Shadow</i> and <i>Married</i> bits is turned on.			

Table 14-3: Summary of Identification Values Of Each BXU

Step	Address (Hex)	Data (Hex)	Action
	FF0450E0	0000003E	Single-word write operation to the <i>QMR</i> register of the PS ₁ BXU employing the "Register Request Using a Physical Address" IAC (IAC access type 0100 _B). This operation sets the <i>Married, Toggle-Primary/Shadow</i> , and <i>Shadow</i> control bits. Note that the write enable for the <i>Shadow</i> and <i>Married</i> bits is turned on.
7.			No action required because the AP-Mask and AP-Match registers are not used in this sample configuration.

Register Summary of the BXUs in the Sample Configuration

Tables 14-3 and 14-4 summarize the final identification values for the various registers. Because some of the registers of the BXU are written in several steps, the final value of the register shown in the register summaries of each BXU may not agree with the values shown in the individual steps previously described. Note that any value shown for a master BXU is identical for its checker BXU. This matching occurs automatically because of two actions: the checker operates in lockstep with the master, and the *INITID* pins of the master and checker pairs are connected together.

Source-ID	Source-ID	Source-ID	Source-ID	Source-ID	Source-ID
0000 0100 _B (4)	0000 0100 _B (4)	0000 0100 _B (4)	0000 0100 _B (4)	0000 0100 _B (4)	0000 0100 _B (4)
0000 0110 _B (6)	0000 0110 _B (6)	0000 0110 _B (6)	0000 0110 _B (6)	0000 0110 _B (6)	0000 0110 _B (6)
0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)

NOTE: The numerical value within the parentheses is the decimal equivalent of the binary number.

Table 14-3: Summary Of Identification Values Of Each BXU on AP-Bus₀

Register	Field	BXU*			
		PP ₀	PS ₀	I/OP ₀	I/OS ₀
Logical-ID	Unit-ID	00 0000 _B (0)	00 0000 _B (0)	00 0001 _B (1)	00 0001 _B (1)
Physical-ID	Component-ID	0 0000 _B (0)	0 0001 _B (1)	0 0010 _B (2)	0 0011 _B (3)
Arbitration-ID	Drive	00 _B (0)	00 _B (0)	00 _B (0)	00 _B (0)
	Count	0000 _B (0)	0000 _B (0)	0001 _B (1)	0001 _B (1)
Module-Error-ID	Confinement-ID	0000 0100 _B (4)	0000 0101 _B (5)	0000 0110 _B (6)	0000 0111 _B (7)
	Source-ID	000 0000 _B (0)	000 0000 _B (0)	000 0000 _B (0)	000 0000 _B (0)
Bus-Error-ID	Confinement-ID	0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)	0000 0000 _B (0)
	Source-ID	000 0100 _B (4)	000 0101 _B (5)	000 0110 _B (6)	000 0111 _B (7)
Spouse-ID	Spouse-ID	0000 0101 _B (5)	0000 0100 _B (4)	0000 0111 _B (7)	0000 0110 _B (6)

NOTE:

* The numerical value within the parentheses is the decimal equivalent of the binary number.

Table 14-4: Summary Of Identification Values Of Each BXU on AP-Bus₁

Register	Field	BXU*			
		PP ₁	PS ₁	I/OP ₁	I/OS ₁
Logical-ID	Unit-ID	00 0000 _B (0)	00 0000 _B (0)	00 0001 _B (1)	00 0001 _B (1)
Physical-ID	Component-ID	0 0000 _B (0)	0 0001 _B (1)	0 0010 _B (2)	0 0011 _B (3)
Arbitration-ID	Drive	00 _B (0)	00 _B (0)	00 _B (0)	00 _B (0)
	Count	0000 _B (0)	0000 _B (0)	0001 _B (1)	0001 _B (1)
Module-Error-ID	Confinement-ID	0000 0100 _B (4)	0000 0101 _B (5)	0000 0110 _B (6)	0000 0111 _B (7)
	Source-ID	000 0001 _B (1)	000 0001 _B (1)	000 0001 _B (1)	000 0001 _B (1)
Bus-Error-ID	Confinement-ID	0000 0001 _B (1)	0000 0001 _B (1)	0000 0001 _B (1)	0000 0001 _B (1)
	Source-ID	000 0100 _B (4)	000 0101 _B (5)	000 0110 _B (6)	000 0111 _B (7)
Spouse-ID	Spouse-ID	0000 0101 _B (5)	0000 0100 _B (4)	0000 0111 _B (7)	0000 0110 _B (6)

NOTE:

* The numerical value within the parentheses is the decimal equivalent of the binary number.

Register Summary of PP₀

The following information summarizes the value of each register in the PP₀ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00000000	No modification from default values.
<i>System-Bus-ID</i>	00000000	No modification from default values.
<i>LBI-Control</i>	00000030	The INIT-RAM memory recognizer is disabled.
<i>Mask₀</i>	FFFC0000	
<i>Match₀</i>	00040001	Enabled with bus recovery.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the <i>Priority</i> field.
<i>Prefetch-Control</i>	00000004	Prefetch functions enabled. User must send a <i>Start</i> command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000000A	<i>Married</i> bit and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000401	The <i>Confinement-ID</i> field is set to a value of four, and the <i>Source-ID</i> field is set to a value of zero.
<i>Bus-Error-ID</i>	00000009	The <i>Confinement-ID</i> field is set to a value of zero, and the <i>Source-ID</i> field is set to value of four.
<i>Error-Log</i>	00000000	No modification from default values.
<i>Error-Record</i>	00??????	No modification from default values.
<i>FRC-Splitting-Control</i>	00000000	No modification from default values.
<i>Spouse-ID</i>	00000500	The <i>Spouse-ID</i> field is set to a value of five.

Register Summary of PS₀

The following information summarizes the value of each register in the PS₀ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00040000	The <i>Component-ID</i> field is set to a value of one.
<i>System-Bus-ID</i>	00000000	No modification from default values.
<i>LBI-Control</i>	00000030	The INIT-RAM memory recognizer is disabled.
<i>Mask₀</i>	FFFC0000	
<i>Match₀</i>	00040001	Enabled with bus recovery.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the <i>Priority</i> field.
<i>Prefetch-Control</i>	00000004	Prefetch functions enabled. User must send a <i>Start</i> command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000001A	<i>Married</i> bit, <i>Shadow</i> bit, and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000500	The <i>Confinement-ID</i> field is set to a value of five, and the <i>Source-ID</i> field is set to a value of zero.
<i>Bus-Error-ID</i>	0000000A	The <i>Confinement-ID</i> field is set to a value of zero, and the <i>Source-ID</i> field is set to a value of five.

Register	Value (Hex)	Comment
<i>Error-Log</i>	00000000	No modification from default values.
<i>Error-Record</i>	00??????	No modification from default values.
<i>FRC-Splitting-Control</i>	00000000	No modification from default values.
<i>Spouse-ID</i>	00000400	The <i>Spouse-ID</i> field is set to a value of four.

Register Summary of I/OP₀

The following information summarizes the value of each register in the I/OP₀ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000001	
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	FFFC0000	
<i>AP-Match</i>	00040001	This register is enabled to recognize addresses 00040000 to 0007FFFF (256KB window).
<i>Logical-ID</i>	00040000	The <i>Unit-ID</i> field is set to a value of one.
<i>Physical-ID</i>	00080000	The <i>Component-ID</i> field is set to a value of two.
<i>System-Bus-ID</i>	00000000	No modification from default values.
<i>LBI-Control</i>	000000F0	The INIT-RAM memory recognizer is disabled and the BXU acts as primary bus master on the L-bus.
<i>Mask₀</i>	00000000	No modification from default values.
<i>Match₀</i>	00000000	No modification from default values.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	This register is not used by the I/O module.
<i>Prefetch-Control</i>	00000000	No modification from default values.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000000A	<i>Married</i> bit and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000600	The <i>Confinement-ID</i> field is set to a value of six, and the <i>Source-ID</i> field is set to a value of zero.
<i>Bus-Error-ID</i>	0000000C	The <i>Confinement-ID</i> field is set to a value of zero, and the <i>Source-ID</i> field is set to a value of six.
<i>Error-Log</i>	00000000	No modification from default values.

Register	Value (Hex)	Comment
Error-Record	00?????	No modification from default values.
FRC-Splitting-Control	00000000	No modification from default values.
Spouse-ID	00000700	The Spouse-ID field is set to a value of seven.
Arbitration-ID	00000001	No modification from default values.
AP-Control	00000000	No modification from default values.
AP-Mask	FFFC0000	No modification from default values.
AP-Match	00040001	No modification from default values.
Logical-ID	00040000	No modification from default values.
Physical-ID	00080000	No modification from default values.
System-Bus-ID	00000000	No modification from default values.
LB1-Control	000000F0	No modification from default values.
Mask	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Private-Mask	00000000	No modification from default values.
Memory-Mask	00000000	No modification from default values.
Private-Mask	00000000	No modification from default values.
Cache-Configuration	00000000	No modification from default values.
Processor-Priority	00000000	No modification from default values.
Fetch-Control	00000000	No modification from default values.
Lock	00000000	No modification from default values.
PTI	00000007	No modification from default values.
FRC	00000000	No modification from default values.
QMR	0000000A	No modification from default values.
PTZ	00000000	No modification from default values.
Maximum	00000000	No modification from default values.
Mobile-Error-ID	00000000	No modification from default values.
Bus-Error-ID	0000000C	No modification from default values.
Error-Log	00000000	No modification from default values.

Register Summary of I/OS₀

The following information summarizes the value of each register in the I/OS₀ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000001	
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	FFFC0000	
<i>AP-Match</i>	00040001	This register is enabled to recognize addresses 00040000 to 0007FFFF (256KB window).
<i>Logical-ID</i>	00040000	The <i>Unit-ID</i> field is set to a value of one.
<i>Physical-ID</i>	000C0000	The <i>Component-ID</i> field is set to a value of three.
<i>System-Bus-ID</i>	00000000	No modification from default values.
<i>LBI-Control</i>	000000F0	The INIT-RAM memory recognizer is disabled and the BXU acts as primary bus master on the L-bus.
<i>Mask₀</i>	00000000	No modification from default values.
<i>Match₀</i>	00000000	No modification from default values.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	This register is not used by the I/O module.
<i>Prefetch-Control</i>	00000000	No modification from default values.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000001A	<i>Married</i> bit, <i>Shadow</i> bit, and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000701	The <i>Confinement-ID</i> field is set to a value of seven, and the <i>Source-ID</i> field is set to a value of zero.

Register	Value (Hex)	Comment
<i>Bus-Error-ID</i>	0000000F	The <i>Confinement-ID</i> field is set to a value of zero, and the <i>Source-ID</i> field is set to a value of seven.
<i>Error-Log</i>	00000000	No modification from default values.
<i>Error-Record</i>	00??????	No modification from default values.
<i>FRC-Splitting-Control</i>	00000000	No modification from default values.
<i>Spouse-ID</i>	00000600	The <i>Spouse-ID</i> field is set to a value of six.

Register Summary of PP₁

The following information summarizes the value of each register in the PP₁ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00000000	No modification from default values.
<i>System-Bus-ID</i>	00004000	
<i>LBI-Control</i>	00000030	The INIT-RAM memory recognizer is disabled.
<i>Mask₀</i>	FFFC0000	
<i>Match₀</i>	00040001	Enabled with bus recovery.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the <i>Priority</i> field.
<i>Prefetch-Control</i>	00000004	Prefetch functions enabled. User must send a <i>Start</i> command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, set to a value of zero for Checker.
<i>QMR</i>	0000000A	<i>Married</i> bit and the <i>Toggle-Primary/Shadow</i> are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000402	The <i>Confinement-ID</i> field is set to a value of four, and the <i>Source-ID</i> field is set to a value of one.
<i>Bus-Error-ID</i>	00000108	The <i>Confinement-ID</i> field is set to a value of one, and the <i>Source-ID</i> field is set to value of four.
<i>Error-Log</i>	00000000	No modification from default values.

Register	Value (Hex)	Comment
<i>Error-Record</i>	00?????	No modification from default values.
<i>FRC-Splitting-Control</i>	00000000	No modification from default values.
<i>Spouse-ID</i>	00000500	The <i>Spouse-ID</i> field is set to a value of five.
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00000000	No modification from default values.
<i>System-Bus-ID</i>	00004000	
<i>LB1-Control</i>	00000030	The INIT-RAM memory recognizer is disabled.
<i>Mask</i>	FF-C0000	
<i>Match</i>	00040001	Enabled with bus recovery.
<i>Mask</i>	00000000	No modification from default values.
<i>Match</i>	00000000	No modification from default values.
<i>Mask</i>	00000000	No modification from default values.
<i>Match</i>	00000000	No modification from default values.
<i>Private-Mask</i>	00000000	No modification from default values.
<i>Private-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the Priority field.
<i>Prefetch-Control</i>	00000004	Prefetch functions enabled. User must set a start command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FTI</i>	00000007	Sets the Wait-for-BERL, the Error-Reporting-Enable, and the Identify-Device-Disable bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, set to a value of zero for Checker.
<i>QMR</i>	0000000A	Married bit and the Toggle-Primary/Shadow are set.
<i>FTZ</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000403	The Confinement-ID field is set to a value of four, and the Source-ID field is set to a value of one.
<i>Bus-Error-ID</i>	00000108	The Confinement-ID field is set to a value of one, and the Source-ID field is set to a value of four.
<i>Error-Log</i>	00000000	No modification from default values.

Register Summary of PS₁

The following information summarizes the value of each register in the PS₁ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00040000	The <i>Component-ID</i> field is set to a value of one.
<i>System-Bus-ID</i>	00004000	
<i>LBI-Control</i>	00000030	The INIT-RAM memory recognizer is disabled.
<i>Mask₀</i>	FFFC0000	
<i>Match₀</i>	00040001	Enabled with bus recovery.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the <i>Priority</i> field.
<i>Prefetch-Control</i>	00000004	Prefetch functions enabled. User must send a <i>Start</i> command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000001A	<i>Married</i> bit, <i>Shadow</i> bit, and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000503	The <i>Confinement-ID</i> field is set to a value of five, and the <i>Source-ID</i> field is set to a value of one.
<i>Bus-Error-ID</i>	0000010B	The <i>Confinement-ID</i> field is set to a value of one, and the <i>Source-ID</i> field is set to value of five.

Register	Value (Hex)	Comment
<i>Error-Log</i>	00000000	No modification from default values.
<i>Error-Record</i>	00??????	No modification from default values.
<i>FRC-Splitting-Control</i>	00000000	No modification from default values.
<i>Spouse-ID</i>	00000400	The <i>Spouse-ID</i> field is set to a value of four.
<i>Arbitration-ID</i>	00000000	No modification from default values.
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	00000000	No modification from default values.
<i>AP-Match</i>	00000000	No modification from default values.
<i>Logical-ID</i>	00000000	No modification from default values.
<i>Physical-ID</i>	00040000	The Component-ID field is set to a value of one.
<i>System-Bus-ID</i>	00040000	
<i>EMI-Control</i>	00000030	The INT-RAM memory recognizer is disabled.
<i>Mask</i>	FFFFC000	
<i>Match</i>	00040001	Enabled with bus recovery.
<i>Mask</i>	00000000	No modification from default values.
<i>Match</i>	00000000	No modification from default values.
<i>Mask</i>	00000000	No modification from default values.
<i>Match</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	User must set the Priority field.
<i>Fetch-Control</i>	00000004	Fetch functions enabled. User must set a Start command to activate channel.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FLU</i>	00000007	Sets the Wait-for-BERR, the Error-Reporting-Enable, and the Identity-Device-Disable bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for a checker.
<i>QMR</i>	0000001A	Master bit, Shadow bit, and the Toggle-Shadow bit are set.
<i>FTZ</i>	00000000	No modification from default values.
<i>Maximum</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000203	The Component-ID field is set to a value of five, and the Source-ID field is set to a value of one.
<i>Bus-Error-ID</i>	0000010B	The Component-ID field is set to a value of one, and the Source-ID field is set to a value of five.

Register Summary of I/OP₁

The following information summarizes the value of each register in the I/OP₁ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000001	
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	FFFC0000	
<i>AP-Match</i>	00040001	This register is enabled to recognize addresses 00040000 to 0007FFFF (256KB window).
<i>Logical-ID</i>	00040000	The <i>Unit-ID</i> field is set to a value of one.
<i>Physical-ID</i>	00080000	The <i>Component-ID</i> field is set to a value of two.
<i>System-Bus-ID</i>	00000000	No modification from default values.
<i>LBI-Control</i>	000000F0	The INIT-RAM memory recognizer is disabled and the BXU acts as primary bus master on the L-bus.
<i>Mask₀</i>	00000000	No modification from default values.
<i>Match₀</i>	00000000	No modification from default values.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	This register is not used by the I/O module.
<i>Prefetch-Control</i>	00000000	No modification from default values.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> bit, the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000000A	<i>Married</i> bit and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000603	The <i>Confinement-ID</i> field is set to a value of six, and the <i>Source-ID</i> field is set to a value of one.
<i>Bus-Error-ID</i>	0000010D	The <i>Confinement-ID</i> field is set to a value of one, and the <i>Source-ID</i> field is set to a value of six.

Register Summary of I/OS₁

The following information summarizes the value of each register in the I/OS₁ BXU.

Register	Value (Hex)	Comment
<i>Arbitration-ID</i>	00000001	
<i>AP-Control</i>	00000000	No modification from default values.
<i>AP-Mask</i>	FFFC0000	
<i>AP-Match</i>	00040001	This register is enabled to recognize addresses 00040000 to 0007FFFF (256KB window).
<i>Logical-ID</i>	00040000	The <i>Unit-ID</i> field is set to a value of one.
<i>Physical-ID</i>	000C0000	The <i>Component-ID</i> field is set to a value of three. }
<i>System-Bus-ID</i>	00040000	
<i>LBI-Control</i>	000000F0	The INIT-RAM memory recognizer is disabled and the BXU acts as primary bus master on the L-bus.
<i>Mask₀</i>	00000000	No modification from default values.
<i>Match₀</i>	00000000	No modification from default values.
<i>Mask₁</i>	00000000	No modification from default values.
<i>Match₁</i>	00000000	No modification from default values.
<i>Mask₂</i>	00000000	No modification from default values.
<i>Match₂</i>	00000000	No modification from default values.
<i>Private-Memory-Mask</i>	00000000	No modification from default values.
<i>Private-Memory-Match</i>	00000000	No modification from default values.
<i>Cache-Configuration</i>	00000000	No modification from default values.
<i>Processor-Priority</i>	00000000	This register is not used by the I/O module.
<i>Prefetch-Control</i>	00000000	No modification from default values.
<i>Lock</i>	00000000	This register is not used in Processor mode.
<i>FT1</i>	00000007	Sets the <i>Wait-for-BERL</i> , the <i>Error-Reporting-Enable</i> , and the <i>Identify-Device-Disable</i> bits.
<i>FRC</i>	0000000M	M is set to a value of one for a master, to a value of zero for Checker.
<i>QMR</i>	0000001A	<i>Married</i> bit, <i>Shadow</i> bit, and the <i>Toggle-Primary/Shadow</i> bit are set.
<i>FT2</i>	00000000	No modification from default values.
<i>Maxtime</i>	00000000	No modification from default values; minimum transient wait selected.
<i>Module-Error-ID</i>	00000702	The <i>Confinement-ID</i> field is set to a value of seven, and the <i>Source-ID</i> field is set to a value of one.
<i>Bus-Error-ID</i>	0000010E	The <i>Confinement-ID</i> field is set to a value of one, and the <i>Source-ID</i> field is set to a value of seven.

Register	Value (Hex)	Comment
Error-Log	00000000	No modification from default values.
Error-Record	00??????	No modification from default values.
FRC-Splitting-Control	00000000	No modification from default values.
Spouse-ID	00000600	The Spouse-ID field is set to a value of six.
Arbitration-ID	00000001	No modification from default values.
AP-Control	00000000	No modification from default values.
AP-Mask	FFFF0000	No modification from default values.
AP-Match	00040001	No modification from default values.
Logical-ID	00040000	No modification from default values.
Physical-ID	000C0000	No modification from default values.
System-Bus-ID	00040000	No modification from default values.
LBI-Control	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Match	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Match	00000000	No modification from default values.
Mask	00000000	No modification from default values.
Match	00000000	No modification from default values.
Private-Mask	00000000	No modification from default values.
Private-Match	00000000	No modification from default values.
Cache-Configuration	00000000	No modification from default values.
Processor-Priority	00000000	No modification from default values.
Prefetch-Control	00000000	No modification from default values.
Lock	00000000	No modification from default values.
FTI	00000007	No modification from default values.
FRC	0000000M	No modification from default values.
QMR	0000001A	No modification from default values.
FTZ	00000000	No modification from default values.
Maxtime	00000000	No modification from default values.
Module-Error-ID	00000702	No modification from default values.
Bus-Error-ID	0000010E	No modification from default values.

SUMMARY

Initialization of the 80960 system consists of three phases: default, identification, and parameterization. After RESET is applied, the registers of the BXU are set to a predetermined value listed in Appendix A for each register. These default values allow a small system to operate without additional initialization. These values can be changed during the identification and parameterization phases. The identification phase is used to assign a *Component-ID* field in the *Physical-ID* register. During the parameterization phase, other registers, such as the *Logical-ID* register can be modified by using the IAC register requests.

The initialization of QMR modules requires assistance of another module or agent. This chapter showed an example of initializing and marrying two FRC pairs using another agent.

*Fault-Tolerant
I/O Considerations*

15

CHAPTER 15 FAULT-TOLERANT I/O CONSIDERATIONS

The CPU and memory modules operate synchronously with the system clock. If the I/O module operates synchronously to the system clock, the same detection mechanisms and recovery techniques that are employed by the CPU and memory modules can also be used by the I/O module. Many I/O systems, however, such as an Ethernet™ controller, or an interrupt controller, function asynchronously to the system clock. This chapter presents some of the considerations for asynchronous fault-tolerant I/O systems, which include the following items:

- Interface to an asynchronous I/O system
- Interrupt handling

OVERVIEW

An I/O system can be duplicated to form a master/checker pair if it can operate synchronously to the system clock and can produce identical responses when it is replicated. If both of these conditions are satisfied, the master/checker pair can be married to another master/checker pair. This marriage forms a QMR I/O module, as shown in Figure 15-1. Thus, synchronous I/O modules can take full advantage of the fault-tolerant facilities of the BXU. The error detection mechanisms, error reporting, and recovery mechanisms described in the previous chapters can be used.

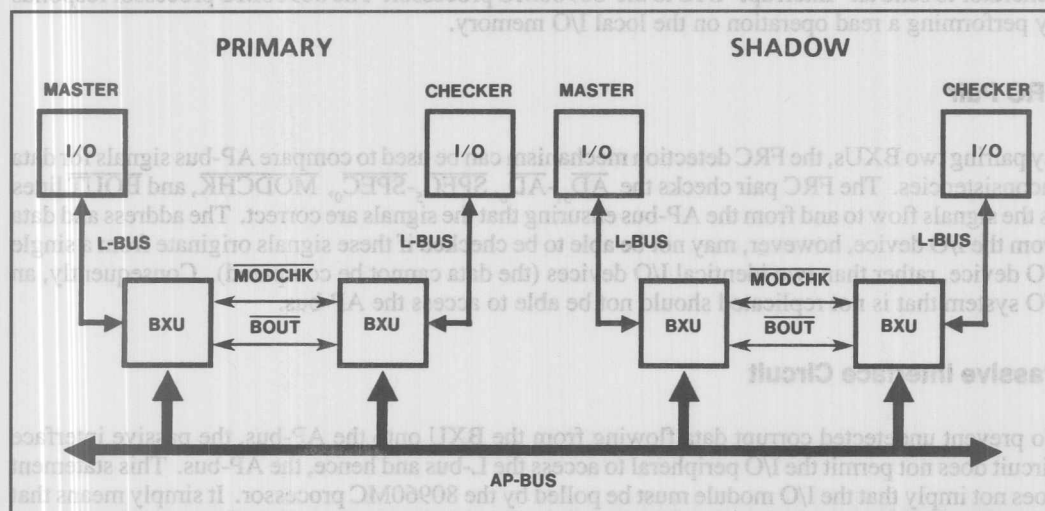


Figure 15-1: Synchronous I/O Interface

Most I/O systems, however, are asynchronous or cannot reproduce identical output signals when replicated. For example, consider an I/O system that contains an analog shaft position. To form an FRC pair, two transducers are used, one for each BXU. For identical shaft positions, the output of

each transducer may vary just enough to cause an error if the outputs were compared. Other examples include an Ethernet™ controller, which cannot guarantee identical output signals, or an interrupt controller where the signals are asynchronous to the system clock.

An I/O system that operates asynchronously or that cannot reproduce identical signals when it is replicated must use a different method to attain fault-tolerant operation. To achieve a fault-tolerant I/O module under these conditions, one design approach may interface an FRC pair to a single I/O system. The FRC pair forms the boundary for the I/O module confinement area to ensure that the data entering the I/O module is error-free. The single I/O system eliminates the problem of similar, but not exact output signals from duplicated peripherals. Thus, the design consists of interfacing two synchronous L-buses to a single asynchronous I/O system. The next section considers the guidelines for designing this type of interface.

FAULT TOLERANT I/O SYSTEM

Figure 15-2 shows an example of an asynchronous I/O interface to a 80960 fault-tolerant system. This system consists of an I/O peripheral, a synchronizer, two passive I/O interface circuits, two comparators, a memory array, two IAC generators, and an FRC master/checker pair. Except for the IAC generator logic and memory array, each of the logic blocks above are described in separate sections. See Chapter 9 for details on the IAC generator.

In a typical sequence of events, the I/O device sends data to the local memory and triggers the IAC generator to send an "Interrupt" IAC to the 80960MC processor. The 80960MC processor responds by performing a read operation on the local I/O memory.

FRC Pair

By pairing two BXUs, the FRC detection mechanism can be used to compare AP-bus signals for data inconsistencies. The FRC pair checks the $AD_{31}-AD_0$, $SPEC_5-SPEC_0$, $MODCHK$, and $BOUT$ lines as the signals flow to and from the AP-bus ensuring that the signals are correct. The address and data from the I/O device, however, may not be able to be checked if these signals originate from a single I/O device, rather than two identical I/O devices (the data cannot be compared). Consequently, an I/O system that is not replicated should not be able to access the AP-bus.

Passive Interface Circuit

To prevent undetected corrupt data flowing from the BXU onto the AP-bus, the passive interface circuit does not permit the I/O peripheral to access the L-bus and hence, the AP-bus. This statement does not imply that the I/O module must be polled by the 80960MC processor. It simply means that any data transmitted to the AP-bus must originate in the synchronous confinement areas of the master and checker, such as an IAC Generator. Two L-buses are used in order to ensure that logic failures on the L-bus are detected. Thus, there is a passive interface circuit for each L-bus.

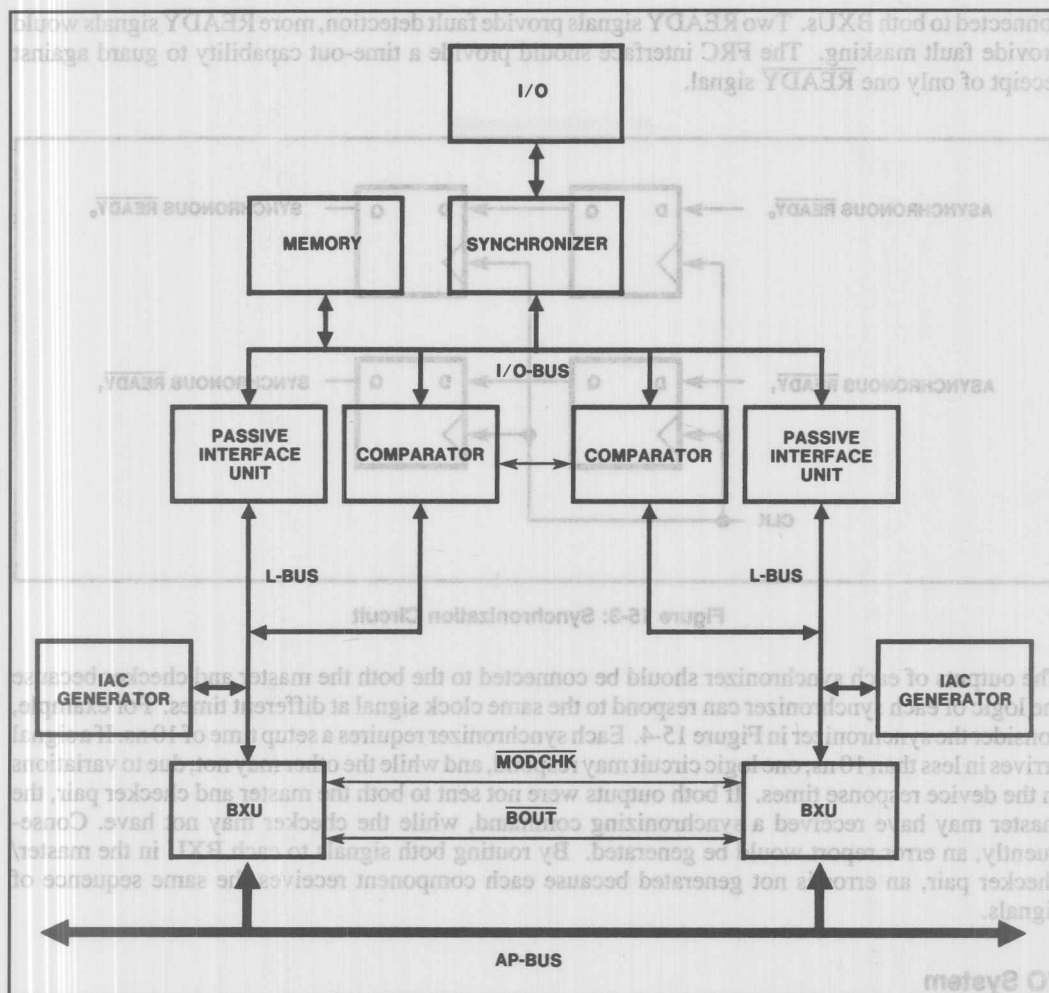


Figure 15-2: Asynchronous I/O Interface

Synchronizer

The synchronizer makes the signal assertions from the I/O device synchronous to the system clock. All of the synchronizer outputs are connected to both the master and checker. Otherwise, FRC failures may occur at the AP-bus interface if the signals arrive at the interface at slightly different times. Synchronizers can be replicated provided that the signal outputs are connected to both the master and checker.

Figure 15-3 shows an example of a synchronizer for the $\overline{\text{READY}}$ signal. The two stages of the synchronization circuit reduces the metastable effects. Both synchronous $\overline{\text{READY}}$ signals must be

connected to both BXUs. Two $\overline{\text{READY}}$ signals provide fault detection, more $\overline{\text{READY}}$ signals would provide fault masking. The FRC interface should provide a time-out capability to guard against receipt of only one $\overline{\text{READY}}$ signal.

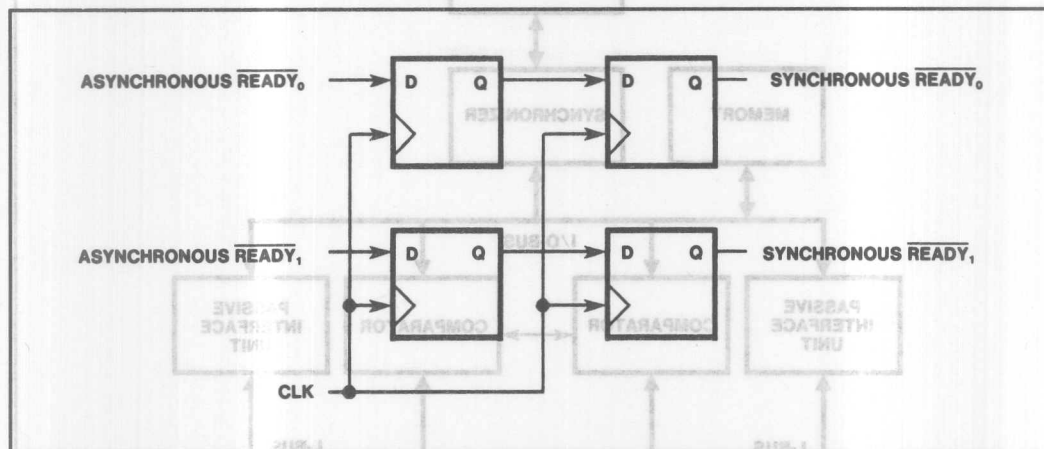


Figure 15-3: Synchronization Circuit

The outputs of each synchronizer should be connected to both the master and checker because the logic of each synchronizer can respond to the same clock signal at different times. For example, consider the synchronizer in Figure 15-4. Each synchronizer requires a setup time of 10 ns. If a signal arrives in less than 10 ns, one logic circuit may respond, and while the other may not, due to variations in the device response times. If both outputs were not sent to both the master and checker pair, the master may have received a synchronizing command, while the checker may not have. Consequently, an error report would be generated. By routing both signals to each BXU in the master/checker pair, an error is not generated because each component receives the same sequence of signals.

I/O System

A fault-tolerant I/O system must ensure two actions: it must not corrupt the rest of the system by sending inaccurate data, and it must perform the intended function (for example, a motor must move the commanded degrees). To comply with the first requirement, FRC pairs can be used to detect any failure on the L-bus or by the BXU in the I/O module. The passive interface and software checks can be used to prevent addresses and data (passively generated by the I/O module) from corrupting the system. The I/O module can be active (initiate requests without being commanded by a processor or other device), if it issues a request from a redundant source. For example, two IAC generators can be located on each L-bus (one generator per bus). In this manner, the I/O system cannot place a data at a random address in which software cannot check.

The second requirement is a system design issue that applies fault-tolerant design techniques to the I/O destination subsystems. Because there is a broad range of I/O systems available, this fault-tolerant design consideration is beyond the scope of this manual.

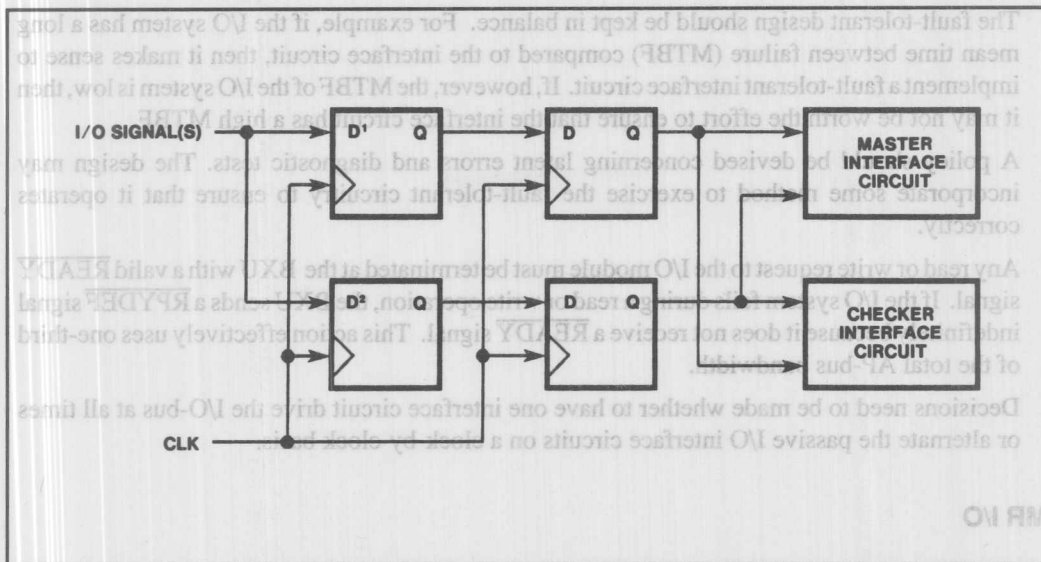


Figure 15-4: Synchronization Circuit Example

Comparators

The comparators perform clock-by-clock checking on every signal passing across the passive interface (this also implies that the passive interface must be synchronous). The comparators themselves need not be fault-tolerant since they are replicated. However, they should have some fault-tolerant way of reporting errors. For example, if a master comparator detects an error caused by the master BXU, and then reports the error to the master BXU, the error report may not be propagated. Errors at the I/O confinement area interface should be reported to both the master and checker to ensure errors get reported in case of an L-bus failure.

The comparator ensures that output data and the control signals are correct. The comparator checks signals on a clock-by-clock basis. Under normal operation, the comparators check the information as it flows to or from the L-bus from or to the I/O-bus. If the I/O system and BXUs place data on each side of the comparator, potential conflicts may occur. These conflicts are resolved by designing communication lines between each comparator, similar to the BOUT line between the FRC BXUs. This connection ensures that when activity on the L-bus and I/O-bus coexist, an error is not generated.

Design Issues

There are several issues that arise when designing a fault-tolerant I/O system. These issues are totally dependent upon the design goals and implementation. Consequently, the following issues are listed to serve as a checklist.

- The fault-tolerant design should be kept in balance. For example, if the I/O system has a long mean time between failure (MTBF) compared to the interface circuit, then it makes sense to implement a fault-tolerant interface circuit. If, however, the MTBF of the I/O system is low, then it may not be worth the effort to ensure that the interface circuit has a high MTBF.
- A policy should be devised concerning latent errors and diagnostic tests. The design may incorporate some method to exercise the fault-tolerant circuitry to ensure that it operates correctly.
- Any read or write request to the I/O module must be terminated at the BXU with a valid READY signal. If the I/O system fails during a read or write operation, the BXU sends a RPYDEF signal indefinitely because it does not receive a READY signal. This action effectively uses one-third of the total AP-bus bandwidth.
- Decisions need to be made whether to have one interface circuit drive the I/O-bus at all times or alternate the passive I/O interface circuits on a clock-by-clock basis.

QMR I/O

The QMR configuration is based upon marrying two identical synchronous FRC pairs by software. Because asynchronous I/O systems cannot be truly configured as FRC pairs, the QMR configuration described in Chapters 10 through 13 cannot be used. Alternative methods are needed to construct a fault-tolerant I/O system using asynchronous peripherals.

Figure 15-5 shows one alternative that implements two FRC pairs each of which interfaces to an I/O system. For this configuration, the two FRC pairs function as independent I/O systems. Software is responsible for the comparison of signals, detection of errors, and recovery from the errors.

To illustrate the use of non-married FRC pairs, Figure 15-6 shows an example of a motor controller with redundant transducers and control switches. The analog shaft position signals from the transducers are applied to the synchronizers. The outputs of the two synchronizers of each FRC pair are connected to both passive interface circuits to ensure that the BXUs remain in lockstep. The passive interface should be designed to be fail-safe (i.e., it automatically opens the switch when a module failure is detected). The serial/parallel configuration of the switches provides FRC checking on the I/O passive outputs (the bus switches must be closed to turn on motor). This setup guards against a failure of a single module because there is a redundant pair of switches to turn off the motor.

The 80960MC processor can either poll the I/O system, or the I/O system can send an "Interrupt" IAC to the 80960MC processor. In either case, the 80960MC can read the data at the I/O system to perform software voting or averaging. Based upon the results, the 80960MC processor issues a command to open or close the motor switches.

HANDLING INTERRUPTS IN FAULT-TOLERANT SYSTEMS

Interrupts within a fault-tolerant system can be categorized into two classes: those generated by devices local to the interrupted processor, such as timers; and those generated by remote devices,

such as sensors. The former are generally synchronized with the system clock, while the latter are inherently asynchronous.

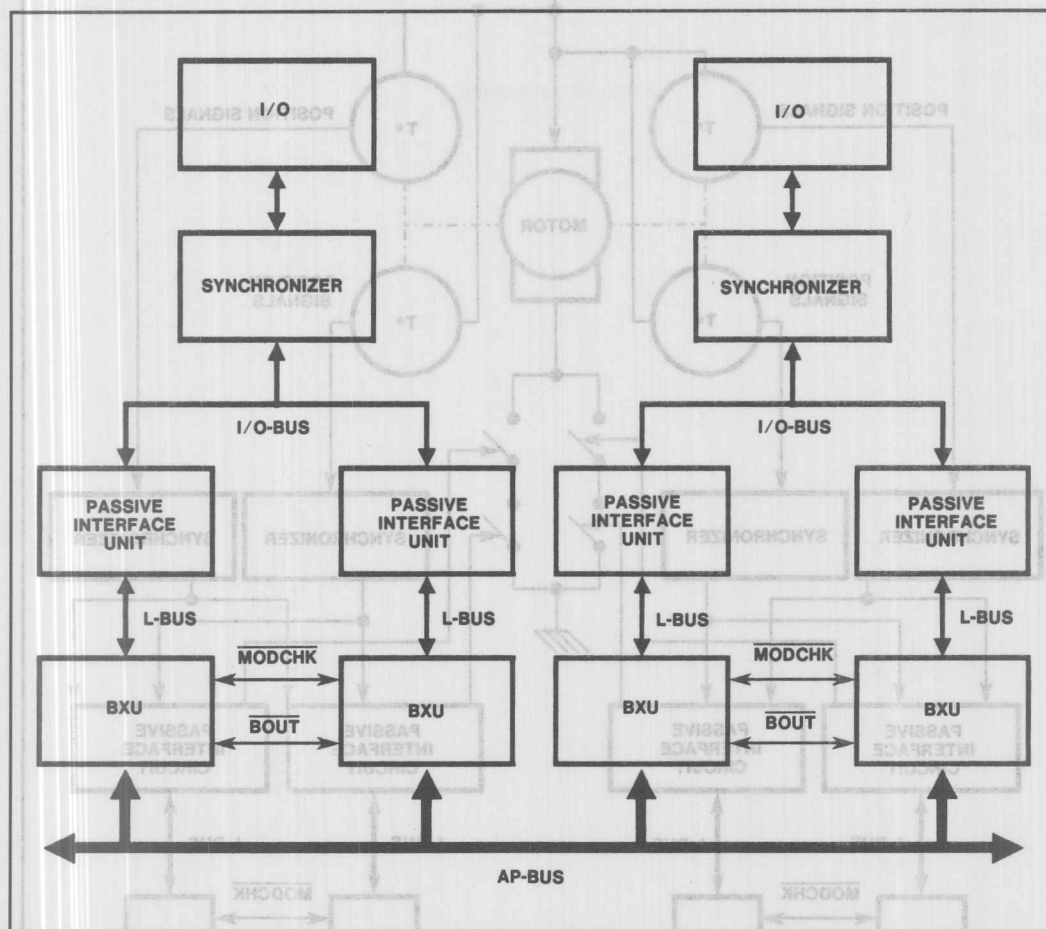


Figure 15-5: Alternative to a QMR I/O System

The synchronous interrupt sources can be replicated and designed to use the on-chip interrupt controllers in each of the replicated processors. This method of handling synchronous interrupts preserves the system's fault tolerance.

Asynchronous interrupt sources, however, cannot be connected to the interrupt controllers of replicated processors without compromising the system's fault tolerance. This is because interrupts arriving at slightly different times could cause the replicated processors to go out of synchronization. Moreover, the differences in setup and response times of the duplicated interrupt logic could cause the processors to go out of synchronization.

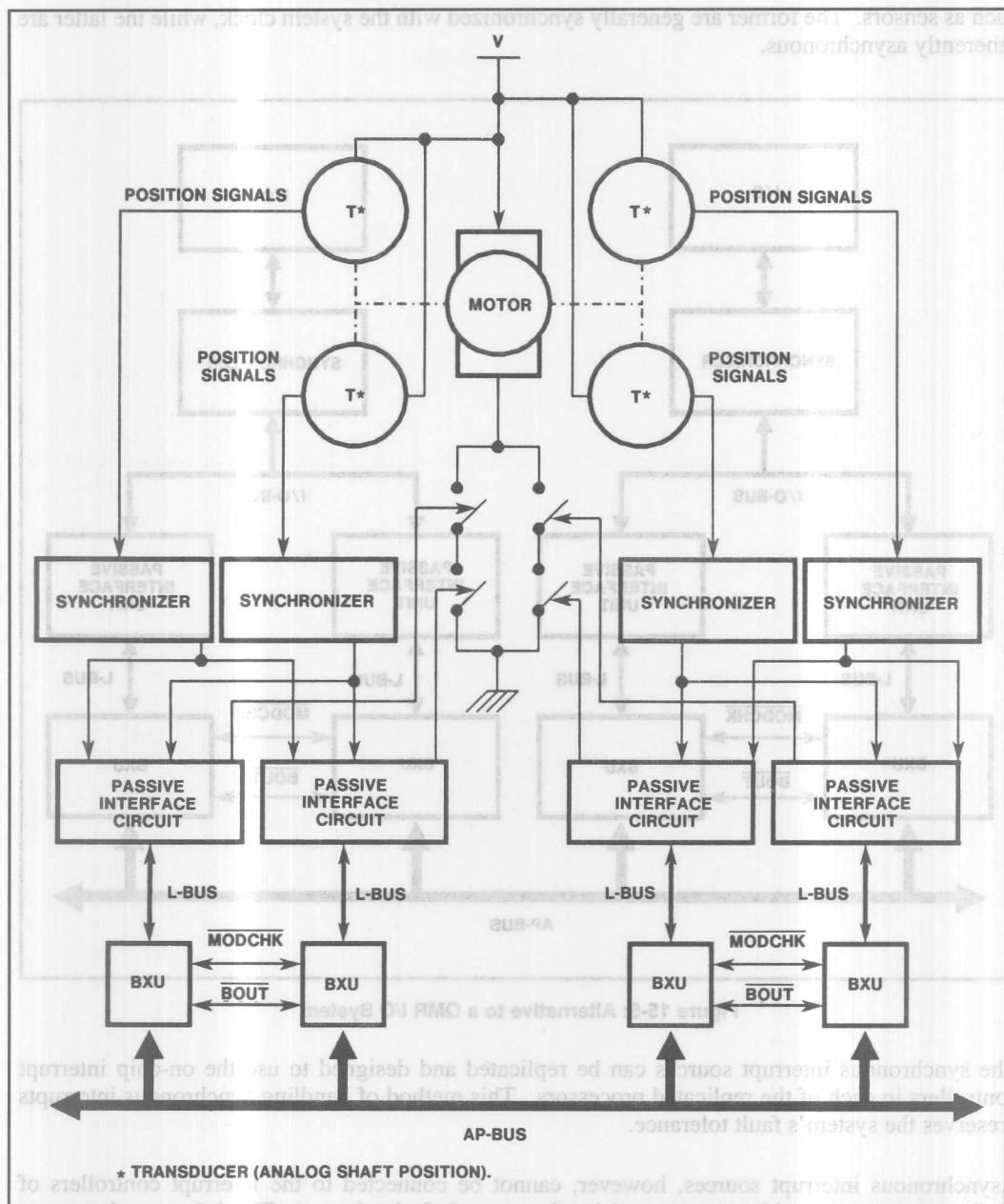


Figure 15-6: I/O Example of Soft QMR I/O System

Consequently, asynchronous interrupts must be isolated from the rest of a fault-tolerant 80960 system. The isolation can be attained by generating the "Interrupt" IAC on the L-bus. This IAC causes the 80960MC processor to respond to the interrupt assuming that it has a higher priority than the current process. Thus, an agent on the I/O subsystem L-bus must accept interrupts and generate their associated IAC messages. This function can be performed by another 80960MC, or by an IAC Generator if the power of the processor is not required (see Chapter 9, "The IAC Generator" section for more details on this circuit).

SUMMARY

The fault-tolerant facilities of the BXU are used for synchronous, reproducible I/O systems. For asynchronous or non-reproducible I/O systems, a fault-tolerant configuration can be built by using non-married FRC pairs to form a redundant I/O system with data comparisons done by software. The FRC pairs check the data to the AP-bus and software routines must ensure that no corrupt data leaves the I/O system. The interface between the I/O system and FRC pair includes a passive circuit and a synchronizer. An active I/O system must guarantee that no corrupt data or addresses are transmitted to the AP-bus. Two FRC pairs can be used to form a redundant I/O system with software performing the error detection and recovery.

Appendix
BXU Registers and Commands

A

A

Appendix BXU Registers and Commands

Table A-1: Summary of BXU Registers and Commands

APPENDIX A BXU REGISTERS AND COMMANDS

The BXU has various programmable registers and commands that provide a flexible interface to the systems software. Table A-1 shows a summary of all the registers and commands in the BXU, and table A-2 provides a memory-map of all the BXU's registers. This appendix provides the details of each register and command, which are listed in alphabetical order (the name of the register or command is listed on the top of the page).

If a software program needs to modify a single register bit, it normally reads the entire contents of the register, modifies the appropriate bit, and writes back the entire contents. This process functions properly as long as hardware does not modify the register bits while software performs this read-modify-write operation. For those register bits that hardware can modify at any time, the BXU provides a *Write-Enable* control bit that protects the specified bit from changing during the read-modify-write operation. In this way, if hardware changes a bit in the register during a read-modify-write operation, the software does not rewrite the protected bit.

The *Write-Enable* is turned on or off by the value of the data word in the write operation. Consequently, during a read operation, the *Write-Enable* bit does not provide data. The data returned in this bit position is equal to the corresponding address bit that was used to access the register request. Thus, bits designated as "don't care" bits may have different values when the register is read using different types of IAC requests. To insure consistent operation, software should mask all bit positions that are "don't care" before using the data.

Typically, the BXU registers are written during initialization and system configuration. Registers can be modified during normal operation, but care must be exercised to maintain correct operation of the BXU. The following lists specify the restrictions on register write operations. See the register definitions in this appendix for descriptions of the contents of the registers and for any effects that occur because of register modifications.

The registers listed below may be written at any time without any restrictions, although the optimum time to write to the *Arbitration-ID* register is when there are no outstanding requests.

- *Arbitration-ID* (write-only)
- *COM*
- *IAC-Message-Buffer*
- *Processor-Priority*
- *Test-Detection*
- *Test-Type*

Table A-1: Summary of BXU Registers and Commands

AP-Bus Interface (2)		L-Bus Interface	
Register	Address	Register	Address
PHYSICAL-ID	040 _H	PHYSICAL-ID (LOCAL)	140 _H
LOGICAL-ID	044 _H	LOGICAL-ID (LOCAL)	144 _H
COMPONENT-SPECIFIER (1)	048 _H	LBI-CONTROL	148 _H
ARBITRATION-ID (1)	048 _H	SYSTEM-BUS-ID	14C _H
COM	04C _H	CACHE-CONFIGURATION	150 _H
AP-CONTROL	050 _H	CACHE-TEST	154 _H
FT1	054 _H	LOCAL-BUS-TEST	158 _H
MAXTIME	058 _H	MATCH ₀	160 _H
FRC-SPLITTING-CONTROL	05C _H	MASK ₀	164 _H
TEST-DETECTION	060 _H	MATCH ₁	168 _H
FRC	064 _H	MASK ₁	16C _H
AP-MATCH	068 _H	MATCH ₂	170 _H
AP-MASK	06C _H	MASK ₂	174 _H
		PRIVATE MEMORY MATCH	178 _H
		PRIVATE MEMORY MASK	17C _H
		Command	Address
		INVALIDATE-CACHE	15C _H
		Memory	
		Register	Address
		LOCK	300 _H
		Externally Mapped Registers (Reserved for Memory Controller)	340 _H -37C _H

NOTES:

- (1) The component specifier (read only) and the Arbitration-ID (write-only) are physically two separate registers that share a common address.
- (2) The AP-Bus interface registers are the only registers that can be addressed using access mode.

Table A-1: Summary of BXU Registers and Commands (cont.)

IAC Message Support				Prefetch	
Register		Address		Register	Address
PROCESSOR-PRIORITY ₀		000 _H		PREFETCH-CONTROL	240 _H
PROCESSOR-MESSAGE ₀		010 _H -01C _H			
PROCESSOR-PRIORITY ₁		020 _H			
PROCESSOR-MESSAGE ₁		030 _H -03C _H			

Prefetch Buffer Registers				Fault Tolerance	
Channel	Buffer	Word	Address	Register	Address
0	0	0	3C0 _H	TEST-TYPE	0C0 _H
0	0	1	3C4 _H	SPOUSE-ID	0C4 _H
0	0	2	3C8 _H	QMR	0E0 _H
0	0	3	3CC _H	MODULE-ERROR-ID	0E4 _H
0	1	0	3D0 _H	BUS-ERROR-ID	0E8 _H
0	1	1	3D4 _H	ERROR-LOG	0EC _H
0	1	2	3D8 _H	ERROR-RECORD	0F0 _H
0	1	3	3DC _H	FT2	0F4 _H
1	0	0	3E0 _H	Command	Address
1	0	1	3E4 _H	TEST-REPORT	104 _H
1	0	2	3E8 _H	PRIMARY-CATASTROPHE	108 _H
1	0	3	3EC _H	SHADOW-CATASTROPHE	10C _H
1	1	0	3F0 _H	TERMINATE-PERMANENT-ERROR-WINDOW	110 _H
1	1	1	3F4 _H	ATTACH-BUS	114 _H
1	1	2	3F8 _H	DETACH-BUS	118 _H
1	1	3	3FC _H	SYNC-REFRESH	11C _H

Table A-2: M80960MC/M82965 Register Map

Register Function	Internal Dest Addr (H) (10 Bit Field)	Register Contents							
		3 1	2 4	2 3	1 6	1 5	0 8	0 7	0 0
Processor Priority 0	000				PPPP				PWVF
*	004								
*	008								
*	00C								
Processor Message 0	010								
Processor Message 0	014								
Processor Message 0	018								
Processor Message 0	01C								
Processor Priority 1	020				PPPP				PWVF
*	024								
*	028								
*	02C								
Processor Message 1	030								
Processor Message 1	034								
Processor Message 1	038								
Processor Message 1	03C								
Physical-ID	040			KCCCCC					
Logical-ID	044			UUUUUU					
Read = Component-Specifier	048 Read					TTT		TTTSSSSS	
Write = Arbitration ID	048 Write							DDCCCC	
Com	04C	CCCCCCCC		CCCCCCCC		CCCCCCCC		CCCCCCCC	
AP-Control	050							WADRRS	
FT1	054							WACWEB	
Maxtime	058							WTMMMH	
FRC-Splitting Control	05C							WMWPSF	
Test Detection	060							WTFRRR	
FRC	064							WRMCTM	
AP-Match	068	BBBBBBBB		BBBBBB				II E	
AP-Mask	06C	MMMMMMMM		MMMMMM				NN	
*	070								
*	074								
*	078								
Test Type	08C				TTTT	TTTTTTTT		TTTTTTTT	
Spouse ID	0C0					SSSSSSSS			
*	0C4								
*	0C8								
*	0DC								
QMR	0E0							PSWSTWME	
Module Error ID	0E4					CCCCCCCC		SSSSSSSP	
Bus Error ID	0E8					000000BB		SSSSSSSP	
Error Log	0EC			EC	CVTTTTTP	CCCCCCCC		SSSSSSSP	
Error Record	0F0				VTTTTTP	CCCCCCCC		SSSSSSSP	
FT2	0F4							S WFCWBR	
*	0F8								
*	100								
Test-Report Command	104								
Primary-Catastrophe Command	108								
Shadow-Catastrophe Command	10C								
Term-Perm Err Wind Command	110								
Attach-Bus Command	114								
Detach-Bus Command	118								
Synch-Refresh Command	11C								

The following registers cannot be written during normal operation. Those registers that are not read-only can only be written when there is an IAC request pending to this BXU. Violating this restriction may cause incorrect or unexpected BXU operations to occur.

- *AP-Match*
- *AP-Mask*
- *Bus-Error-ID*
- *Cache-Configuration*
- *Component-Specifier* (read-only)
- *Error-Log*
- *Error-Record*
- *FRC-Split-Control*
- *FT1*
- *LBI-Control*
- *Local-Match*
- *Local-Mask*
- *Logical-ID*
- *Module-Error-ID*
- *Physical-ID*
- *Prefetch-Control*
- *System-Bus-ID* (read-only)

The final set of registers consists of special cases. Most of these registers contain bits that may be modified and other bits that may not be changed during normal operation. The bits that cannot be written during normal operation are either protected with *Write-Enable* control bits or not writable by software (except for the *Bus-Switch-Disable* bit in the *AP-Control* register which is not protected by the *Write-Enable* control bit). The parentheses below enclose the control bits or fields that can be changed during normal operation.

- *AP-Control* (only the *COM-Altered* bit)
- *FRC* (only the *Toggle-M/C* and *Access-Master/Checker* bits)
- *FT2* (only the *Force-Correct* bit)
- *Lock* (only the *Lock-Timed-Out* bit)
- *Maxtime* (only the *Maxtime* bits for on-line repair)
- *QMR* (only the *Toggle-P/S* and *Access-P/S* bits)
- *Spouse-ID* (only *Spouse-ID* field when not married)

The access mode of a Logical Address IAC can only be used to address the AP-bus registers.

The address notation utilized throughout this appendix is the hexadecimal representation of the *Internal Destination* field (bits 0-9) of a Register Request IAC.

This register is the general control and status register of the BXU's AP-bus interface. This register may be written during normal operation of the BXU. Figure A-1 describes the contents of the register.

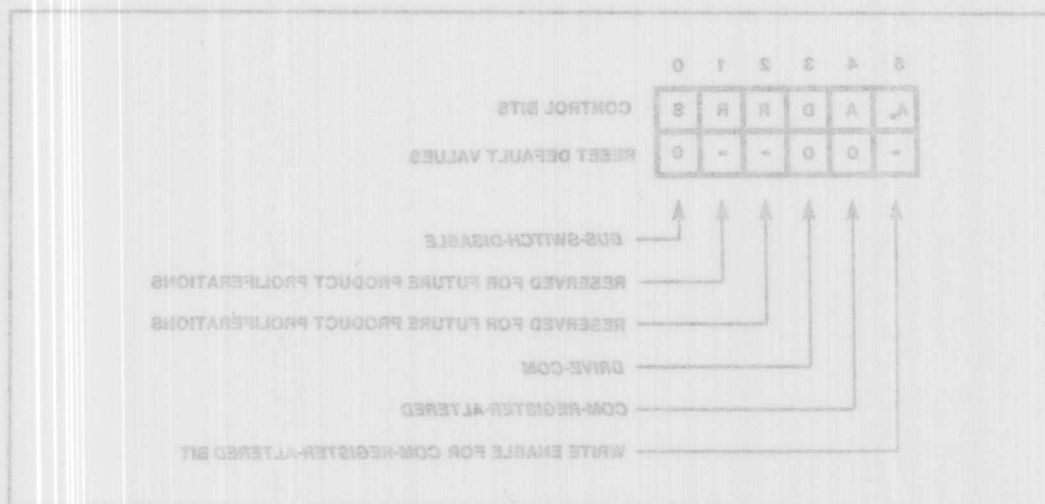


Figure A-1: AP-Control Register Contents

Bit Descriptions

Bus-Switch-Disable

1 = Setting this bit to a value of one disables the bus switching function. The RPYDEF signal resets the bus time-out and places the request currently at the front of the bus pipeline at the end of the pipeline.

0 = Setting this bit to a value of zero enables the bus switching function. The RPYDEF signal is used only to disable the bus time-out. No reordering of the bus pipeline occurs.

Drive-Com

1 = Software may set this bit to force the COM pin to be asserted.

0 = The COM pin is not asserted (if the COM pin is in the P22 register is set, the BXU drives the COM pin low regardless of this setting).

This bit defaults to a value of zero.

Descriptions of the Registers and Commands

AP-Control Register (Address 050_H)

This register is the general control and status register of the BXU's AP-bus interface. This register may be written during normal operation of the BXU. Figure A-1 describes the contents of the register bits.

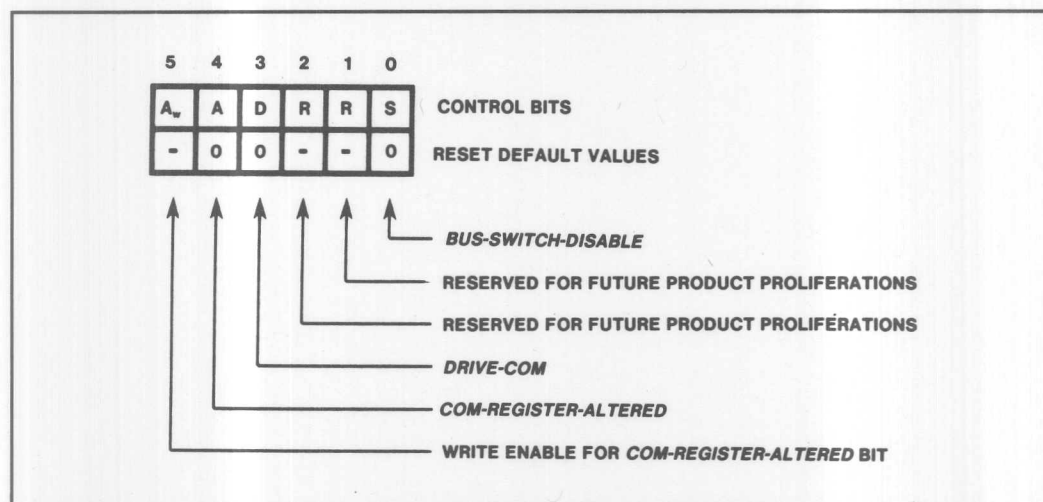


Figure A-1: AP-Control Register Contents

Bit Descriptions

Bus-Switch-Disable

1 = Setting this bit to a value of one disables the bus switching function. The RPYDEF signal resets the bus time-out and places the request currently at the front of the bus pipeline at the end of the pipeline.

0 = Setting this bit to a value of zero enables the bus switching function. The RPYDEF signal is used only to disable the bus time-out. No reordering of the bus pipeline occurs.

Drive-Com

1 = Software may set this bit to force the \overline{COM} pin to be asserted.

0 = The \overline{COM} pin is not asserted (if the Faulty bit in the FT2 register is set, the BXU drives the \overline{COM} pin low regardless of this setting).

This bit defaults to a value of zero.

COM-Register-Altered

1 = This setting indicates that the contents of the *COM* register has changed from the time the *COM-Register-Altered* bit was last cleared.

0 = This setting indicates that the *COM* register has not changed. This bit defaults to a value of zero.

Write Enable for

COM-Register-Altered Bit

1 = This setting allows a write operation to the *COM-Register-Altered* bit.

0 = This setting does not allow a write operation to alter the *COM-Register-Altered* bit.

Figure A-2 describes the contents of the register fields and bits.

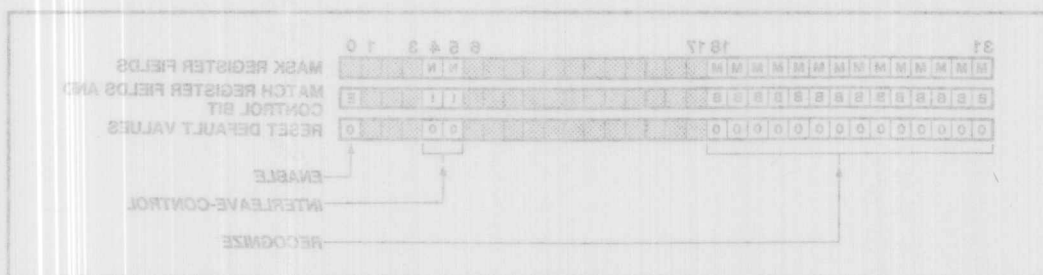


Figure A-2: AP-Mask and AP-Match Register Contents

Field and Control Bit Descriptions

Recognize
For the AP-Mask register, each bit (M) in this field that is set causes the AP-bus address bit to be compared against the corresponding AP-Match register bit. If a bit is cleared, then that bit position is a "don't care" during address recognition.

Interleave-Control
For the AP-Match register, each bit (B) that is set in this field is compared against the corresponding bits in AP-bus address cycles. Thus, these bits provide an address for the partition of memory that is recognized by this address recognizer.

Interleave-Control
These two bits (N) determine the interleaving factor and matching for recognizer in the AP-bus interface. Table A-3 shows the impact of the different configurations of these bits.

AP-Mask and AP-Match Registers (Address 06C_H and 068_H)

The bits in the *AP-Match* register are compared against the corresponding bits in the AP-bus address cycle, and determine which partition of the address space is recognized by an individual BXU. The *AP-Match* register is also used with the *AP-Mask* register to control the interleave factor. The *AP-Mask* and *AP-Match* registers create an AP-bus address recognizer.

If a bit in the *AP-Mask* register is cleared, it causes the corresponding bit position in the *AP-Match* register to be ignored during comparisons.

The *AP-Mask* register should always be setup before the *AP-Match* register is enabled. The *AP-Mask* register defaults to a register value of zero. If the *AP-Mask* register is not changed before the *AP-Match* register is enabled, then the BXU matches on all AP-bus requests (memory and IAC requests).

Figure A-2 describes the contents of the register fields and bits.

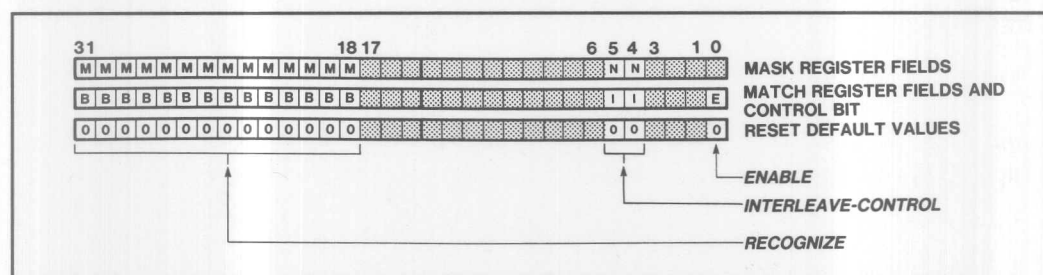


Figure A-2: *AP-Mask* and *AP-Match* Register Contents

Field and Control Bit Descriptions

Recognize

For the *AP-Mask* register, each bit(M) in this field that is set causes the AP-bus address bit to be compared against the corresponding *AP-Match* register bit. If a bit is cleared, then that bit position is a “don’t care” during address recognition.

For the *AP-Match* register, each bit(B) that is set in this field is compared against the corresponding bits in AP-bus address cycles. Thus, these bits provide an address for the partition of memory that is recognized by this address recognizer.

Interleave-Control

These two bits(N) determine the interleaving factor and matching for recognizer in the AP-bus Interface. Table A-3 shows the impact of the different configurations of these bits.

Table A-3: Interleave-Control Bit Settings for Different Configurations

Mask Bits	Match Bits*	AD ₆ , AD ₄ Required for Match*
00	xx	xx (no interleaving)
01	x0	x0 (2 way interleaving)
01	x1	x1 (2 way interleaving)
10†	xx	xx
11	00	00 (4 way interleaving)
11	01	01 (4 way interleaving)
11	10	10 (4 way interleaving)
11	11	11 (4 way interleaving)

NOTES:

* "x" designates a "don't care" value.

† illegal

Enable Bit
(Match Register Only)

1 = Both the *AP-Mask* and *AP-Match* registers are enabled if the *AP-Inactive* bit in the *FT1* register and the *Faulty* bit in the *FT2* register are both zero. If either of these bits is a one, then the *AP-Mask* and *AP-Match* registers are disabled independent of the state of this enable bit.

0 = This address recognizer is disabled.

This bit can also be set by the fault tolerance logic as a result of receiving a *Sync-Refresh* error report.

When the BXU needs to issue a request on the AP-bus, it must actively arbitrate for the bus. The way in which it arbitrates is determined by the contents of this register.

The 80960MC processor can only write to this register (a read operation to this address reads the contents of the *Component-Specifier* register). Since only write operations are performed to this register, configuration software should establish a relationship between the contents of the *Arbitration-ID* register and some other readable register (for example, the *Logical-ID* register). Figure A-3 describes the contents of the register fields.

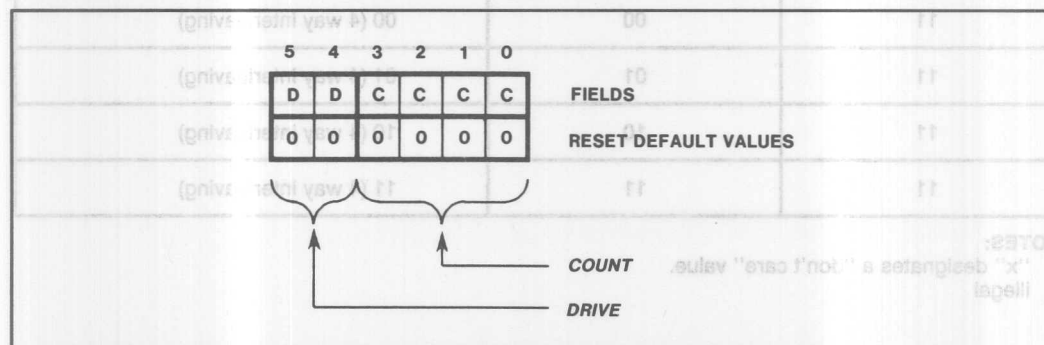


Figure A-3: Arbitration-ID Register Contents

Field Descriptions

Count

The *Count* field determines how many time-step intervals \overline{ARB}_3 is asserted during arbitration. The *Count* field also determines the time-step interval in which the BXU asserts the arbitration line specified by the *Drive* field (see *Drive* field description below). The default value for the *Count* field is fixed in each component. The *Count* field is loaded by an Identify Device Order with bits 0 through 3 of the second data word. The *Count* field defaults to a value of zero at RESET.

Drive

The *Drive* field determines which arbitration line (\overline{ARB}_2 , \overline{ARB}_1 , or \overline{ARB}_0) to assert during the time-step interval specified by the *Count* field. The *Drive* field indicates the agent's priority in that time-step interval: the \overline{ARB}_0 line has the highest priority, followed by \overline{ARB}_1 , then \overline{ARB}_2 . The mapping of the arbitration lines is shown in Table A-4. The *Drive* field defaults to a value of zero at RESET.

Table A-4: Mapping of ARB_3 - ARB_0 to the *Drive* Bits

Drive Field		Arbitration Line
0	0	$\overline{ARB_0}$
0	1	$\overline{ARB_1}$
1	0	$\overline{ARB_2}$
1	1	$\overline{ARB_3}$

The default value for the *Drive* field is set for each component. This field is loaded by an Identify Device command with bits 4 and 5 of the second data word.

Attach-Bus Command (Address 114_H)

This command causes the identified bus to be attached to the system and become active. A write operation to this register location causes the BXU to generate an *Attach-Bus* error report, using its *Bus-Error-ID* register for the *Location-ID* field. The data word sent with the write request is ignored. Before sending this command, software should set the *Sequence* bits in the *Match* registers of the BXU and disable the prefetch function.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with the value of one in the *COM* register.

Bus-Error-ID Register (Address OE8_H)

This register provides the *Location-ID* field for error reports that originate in an AP-bus confinement area. The *Location-ID* field is comprised of the *Source-ID* and the *Confinement-ID* fields of this register. Figure A-4 describes the data fields and the control bits of this register. The *Confinement-ID* field is used for the error report message when the error occurs on the bus of this BXU. The *Confinement-ID* field, however, cannot be read or written by means of this register. The *Confinement-ID* may be changed by modifying the *System-Bus-ID* register.

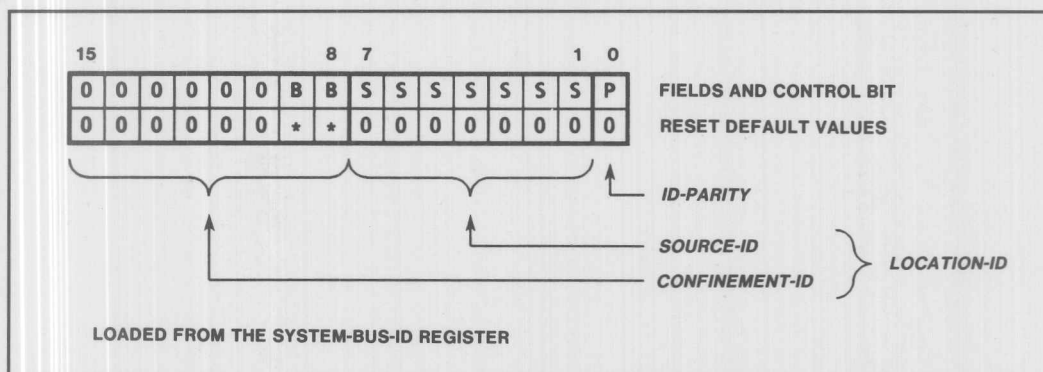


Figure A-4: *Bus-Error-ID* Register Contents

Field and Control Bit Descriptions

ID-Parity Bit

This bit provides even parity for the *Source-ID* and *Confinement-ID* fields in this register. When software loads new values into these fields, it must also load the *ID-Parity* bit to ensure that there are an even number of bits with the value of one in this 16-bit register.

Source-ID

This 7-bit field uniquely identifies a BXU within a confinement area. Software must assign a value to this field. Software uses the *Source-ID* field in the *Error-Log* register to specifically identify which BXU (or Master/Checker pair) had issued the error report message.

Confinement-ID

This 8-bit field is not directly a part of the *Bus-Error-ID* register because the field cannot be read or written using this register address. The *Confinement-ID* field uniquely identifies a confinement area within the system. The BXUs use this value in the error report messages to determine which recovery actions to take. This register is used when the BXU reports an error in its bus confinement area.

original AP-bus confinement ID fields of this register. The Confinement-ID field of this register occurs on the bus of this BXU. The Confinement-ID field of the register. The Confinement-ID field of the register.

Bits 8 and 9 of this register are loaded from the *AP-Bus-ID* field in the *System-Bus-ID* register at RESET. Thus, this field has the correct identification value at all times. The *ID-Parity* bit, however, may not be correct at RESET time. This means that the BXUs should not send any error reports until the parity bit is set correctly by software (the BXUs are able to respond correctly to error report messages). This is important software consideration for the initialization processor that boots the rest of the system.)

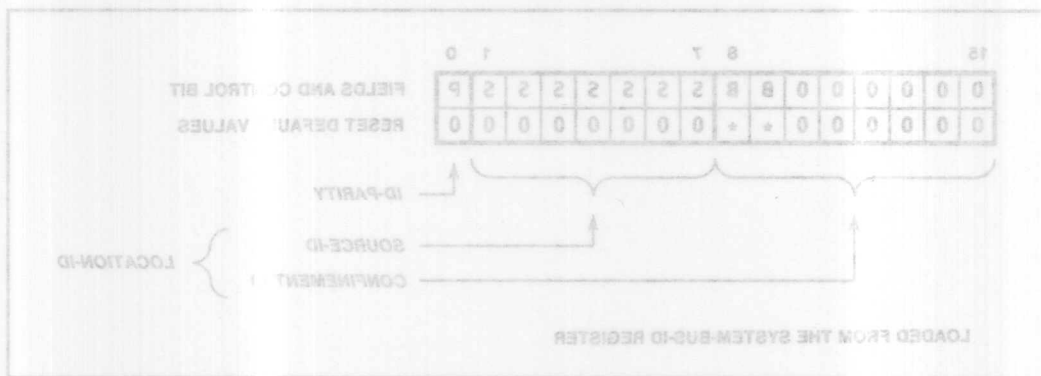


Figure A-4: Bus-Error-ID Register Contents

Field and Control Bit Descriptions

This bit provides even parity for the source-ID and Confinement-ID fields in this register. When software loads new values into these fields, it must also load the ID-Parity bit to ensure that there are an even number of bits with a value of one in this ID-bit register.

ID-Parity Bit

This 7-bit field uniquely identifies a BXU within a confinement area. Software must assign a value to this field. Software uses the source-ID field in the Error-ID register to specifically identify which BXU (or Master/Checker pair) has issued the error report message.

Source-ID

This 8-bit field is not directly a part of the Bus-Error-ID register because the field cannot be read or written using this register address. The Confinement-ID field uniquely identifies a confinement area within the system. The BXUs use this value in the error report messages to determine which recovery actions to take. This register is used when the BXU reports an error in its bus confinement area.

Confinement-ID

Cache-Configuration Register (Address 150_H)

The control bits in this register together with the control bits of the *LBI-Control* register determine the cache configuration. The *Cache-Configuration* register specifies the directory organization and timing options; the *LBI-Control* register specifies the interleaving factor. Figure A-5 describes the contents of the register fields and control bits.

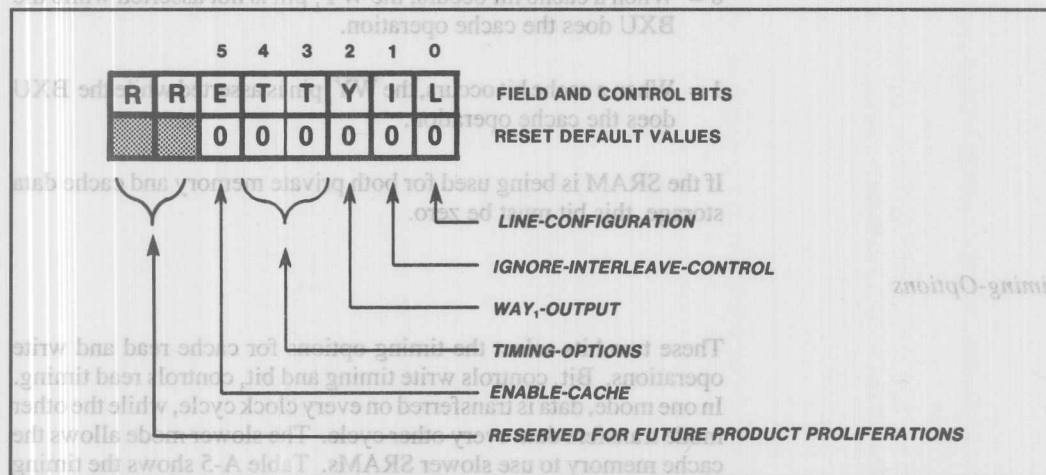


Figure A-5: Cache-Configuration Register Contents

Field and Control Bit Descriptions

Line-Configuration Bit

0	This setting specifies four lines per address block.
1	This setting specifies eight lines per address block.

Ignore-Interleave-Control Bit

0	This setting indicates that the interleave control bits from the <i>LBI-Control</i> register are used.
---	--

1	This setting indicates that the <i>LBI-Control</i> register interleaving bits are ignored. The cache operates in the non-interleaved configuration.
---	---

WAY₁-Output Bit

0 = When a cache hit occurs, the $\overline{\text{WY}}_1$ pin is not asserted while the BXU does the cache operation.

1 = When a cache hit occurs, the $\overline{\text{WY}}_1$ pin is asserted while the BXU does the cache operation.

If the SRAM is being used for both private memory and cache data storage, this bit must be zero.

Timing-Options

These two bits select the timing options for cache read and write operations. Bit₄ controls write timing and bit₃ controls read timing. In one mode, data is transferred on every clock cycle, while the other mode transfers data every other cycle. The slower mode allows the cache memory to use slower SRAMs. Table A-5 shows the timing selections.

Table A-5: Timing Selections

Timing-Option Bits		Speed Selection
Bit ₄	Bit ₃	
1	1	3/6 write timing, 3/6 read timing (fast/fast)
1	0	3/6 write timing, 4/10 read timing (fast/slow)
0	1	4/10 write timing, 3/6 read timing (slow/fast)
0	0	4/10 write timing, 4/10 read timing (slow/slow)

Enable-Cache Bit

1 = This setting enables the cache for operation. Other devices on the L-bus must now insert at least one wait state in their replies. With one wait state the BXU is guaranteed to keep up with the worst case coherency traffic on the AP-bus and L-bus.

0 = This setting disables the cache for operation.

Table A-6 shows the possible cache configurations based upon the Cache Configuration and LBI Control registers.

Table A-6: BXU Cache Configurations

Configuration Number	Size	Lines Per Block	Ways	Sets	BXU Interleave	Cache Configuration Register		LBI Control Register	
						I	L	N	N
1	64K	8	2	64	4	0	1	1	1
2	32K	8	2	64	2	0	1	0	1
3	16K	8	2	64	1	0	1	x	0
						1	1	x	x
4	32K	4	2	64	4	0	0	1	1
5	16K	4	2	64	2	0	0	0	1

Table A-7 shows the address line mapping in the different configurations. The BXU directory logic uses the *tag*, *set*, and *line* bits as part of its directory lookup operation. The interleave bits are not used. Thus, configurations without interleaving are restricted to operation in single AP-bus systems, or dual AP-bus systems where the BXUs on the different buses have the \overline{WY}_1 bit set to different values. The *tag* field is 19 bits long (bit₃₁ through bit₁₃).

Table A-7: Address Mapping for Different Configurations

Configuration	Tag	Set	Line
1	LAD ₃₁ -LAD ₁₃	LAD ₁₄ -LAD ₉	LAD ₈ -LAD ₆
2	LAD ₃₁ -LAD ₁₃	LAD ₁₃ -LAD ₈	LAD ₇ -LAD ₅
3	LAD ₃₁ -LAD ₁₃	LAD ₁₂ -LAD ₇	LAD ₆ -LAD ₄
4	LAD ₃₁ -LAD ₁₃	LAD ₁₃ -LAD ₈	LAD ₇ -LAD ₆
5	LAD ₃₁ -LAD ₁₃	LAD ₁₂ -LAD ₇	LAD ₆ -LAD ₅

Table A-8 shows the SRAM address lines. The \overline{WY}_1 - \overline{WY}_0 and \overline{WD}_1 - \overline{WD}_0 address bits are provided by the BXU. All other address bits are from the L-bus signals.

Table A-8: SRAM Address Lines


CONFIGURATION NUMBER	ADDRESS LINE																																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1																	Y	A	A	A	A	A	A	A	A	A	A	A	R	R			
2,4																		Y	A	A	A	A	A	A	A	A	A	A	A	R	R		
3,5																			Y	A	A	A	A	A	A	A	A	A	A	R	R		
2,4 + PM																	Y	Y	A	A	A	A	A	A	A	A	A	A	A	R	R		
3,5 + PM																	Y	Y	A	A	A	A	A	A	A	A	A	A	A	R	R		

Notes:

A — L-Bus Address

R — Word Bits

Y — Way Bits

 — No Connection

PM — Private Memory

Table A-7 shows the address line mapping in the different configurations. The BXU directory logic uses the tag set and line bits as part of its directory lookup operation. The interleave bits are not used. Thus, configurations without interleaving are restricted to operation in single AP-bus systems, or dual AP-bus systems where the BXUs on the different buses have the \overline{WY}_1 bit set to different values. The tag field is 19 bits long (bit₁₈ through bit₀).

Table A-7: Address Mapping for Different Configurations

Configuration	Tag	Set	Line
1	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀
2	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀
3	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀
4	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀
5	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀	LAD ₁₈ -LAD ₀

Cache-Test Register (Address 154_H)

This register facilitates the testing of the cache directory and control logic of the BXU. Figure A-6 describes the contents of the control bits.

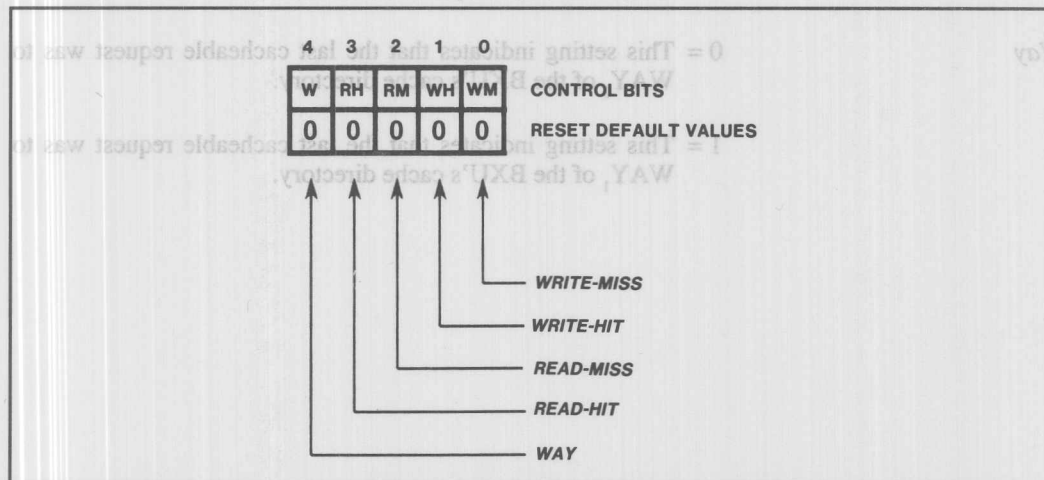


Figure A-6: Cache-Test Register Contents

Bit Descriptions

Write-Miss

0 = This setting indicates that the last cacheable request to this BXU was not a write request that missed the cache.

1 = This setting indicates that the last cacheable request to this BXU was a write request that missed the cache.

Write-Hit

0 = This setting indicates that the last cacheable request to this BXU was not a write request that hit the cache.

1 = This setting indicates that the last cacheable request to this BXU was a write request that hit the cache.

Read-Miss

0 = This setting indicates that the last cacheable request to this BXU was not a read request that missed the cache.

1 = This setting indicates that the last cacheable request to this BXU was a read request that missed the cache.

1 = This setting indicates that the last cacheable request to this BXU was a read request that hit the cache.

Way

0 = This setting indicates that the last cacheable request was to WAY₀ of the BXU's cache directory.

1 = This setting indicates that the last cacheable request was to WAY₁ of the BXU's cache directory.

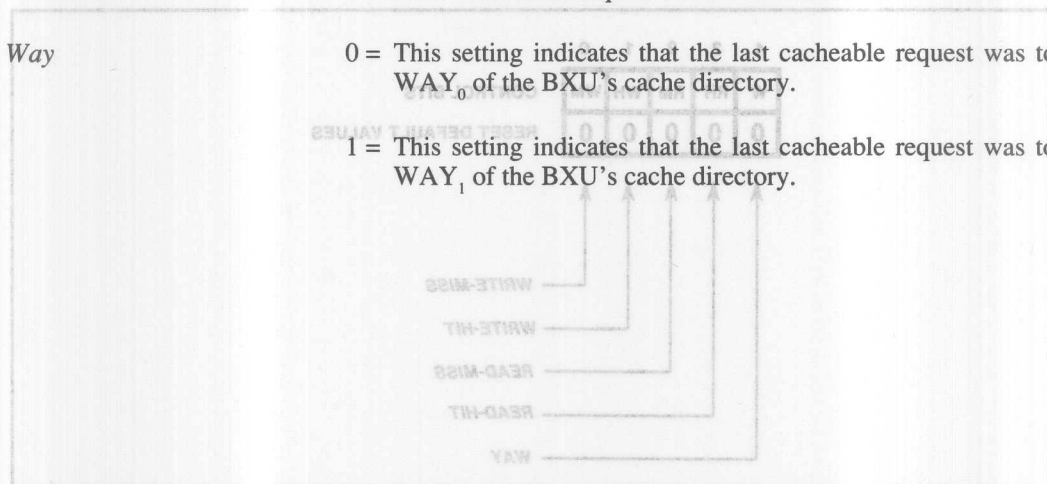


Figure A-8: Cache-Tag Register Contents

Bit Descriptions

0 = This setting indicates that the last cacheable request to this BXU was not a write request that missed the cache.

1 = This setting indicates that the last cacheable request to this BXU was a write request that missed the cache.

0 = This setting indicates that the last cacheable request to this BXU was not a write request that hit the cache.

1 = This setting indicates that the last cacheable request to this BXU was a write request that hit the cache.

0 = This setting indicates that the last cacheable request to this BXU was not a read request that missed the cache.

1 = This setting indicates that the last cacheable request to this BXU was a read request that missed the cache.

Write-Miss

Write-Hit

Read-Miss

COM Register (Address 04C_H)

This register is used for loading external information, such as the type of board the BXU resides on. The register is useful for both initialization and diagnostics.

The *COM* register is actually 38 bits wide. Thirty-two bits are visible and 6 bits are hidden. Write operations to this register clear the hidden bits. The hidden bits are part of a shift chain used for testing the FRC logic in the AP-bus interface. The serial protocol is discussed in Chapter 14. Figure A-7 describes the contents of the register fields.

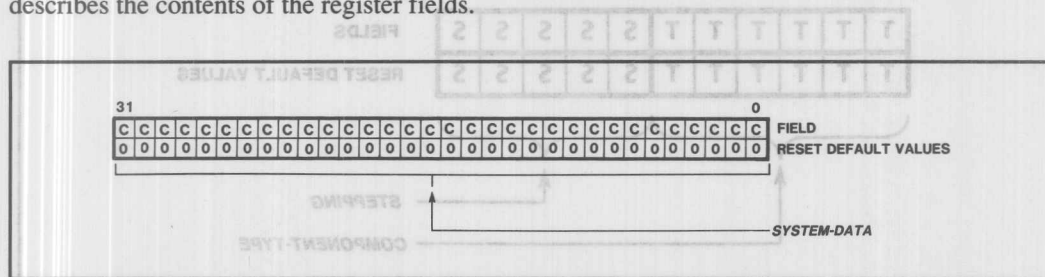


Figure A-7: COM Register Contents

Field Descriptions

System-Data This register is used for various testing and parameterization functions, as defined in Chapter 14. The default for this register is a value of zero.

Component-Specifier Register (Address $_{048H}$)

The contents of this read-only register are fixed at manufacturing and specify the type and stepping of the component. A write operation to this register modifies the *Arbitration-ID* register. Figure A-8 describes the contents of the register fields.

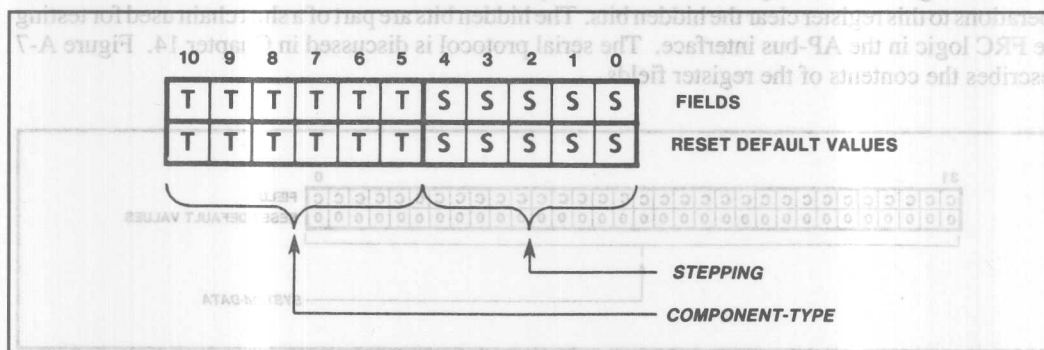


Figure A-8: Component-Specifier Register Contents

Field Descriptions

Stepping

This 5-bit field identifies the stepping number of the component, which is set during manufacturing.

Component-Type

This 6-bit field is set during manufacturing and each BXU is assigned the same value. This field will be used to distinguish a BXU from other future AP-bus components.

Detach-Bus Command (Address 118_H)

This command causes the identified bus to be detached from the system and become inactive. A write operation to this register location causes the BXU to generate *Detach-Bus* error report on the serial error reporting network. The data sent in the data word of the write request is ignored.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with the value of one in the *COM* register.

Figure A-9 describes the contents of the register fields and control bits.

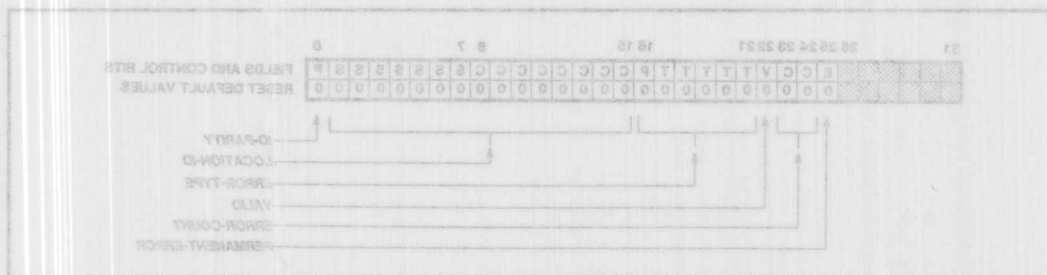


Figure A-9: Error Log Register Contents

Field and Control Bit Descriptions

ID-Parity Bit	This field is identical to the ID-Parity field in the error report message.
Location-ID	This field is identical to the Confirmed-ID and Source-ID fields in the error report message.
Error-Type	This field is identical to the error type field in the error report message. The parity bit, which is generated by the BXU, provides odd parity over the Error-Type field.
Valid Bit	0 = This setting indicates that the other fields in this register are invalid and should be ignored. This bit is set to zero if all error messages received in the last error report were invalid. Recovery does not proceed on error reports that do not have the Valid bit set. 1 = This setting indicates that the other fields in this register are valid.

Error-Log Register(Address OEC_H)

This register records the type of the most recent error report received and the number of errors that have occurred since the last *Terminate-Permanent-Error-Window* command. This register is the primary form of communications between the hardware fault handling logic and the software. Because hardware reads and writes to this register, software should not write to it during normal system operation. If software does write to this register with the *Valid* bit not set, then recovery mechanism 1 is invoked (see Chapter 12 for information on the recovery mechanism 1). This means that a subsequent error results in an error reporting error since recovery mechanism 1 splits the BERL₁-BERL₀ lines.

Figure A-9 describes the contents of the register fields and control bits.

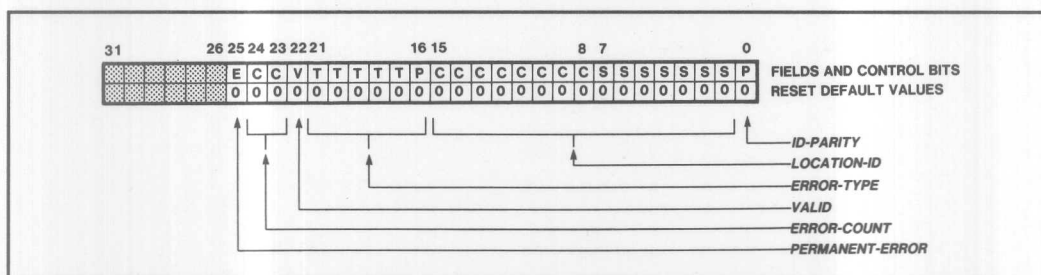


Figure A-9: Errir Log Register Contents

Field and Control Bit Descriptions

ID-Parity Bit

This field is identical to the *ID-Parity* field in the error report message.

Location-ID

This field is identical to the *Confinement-ID* and *Source-ID* fields in the error report message.

Error-Type

This field is identical to the error type field in the error report message. The parity bit, which is generated by the BXU, provides odd parity over the *Error-Type* field.

Valid Bit

0 = This setting indicates that the other fields in this register are invalid and should be ignored. This bit is set to zero if all error messages received in the last error report were invalid. Recovery does not proceed on error reports that do not have the *Valid* bit set.

1 = This setting indicates that the other fields in this register are valid.

If the *Error-Log* register is written by software, the *Valid* bit should be set or recovery mechanism 1 is invoked (see Chapter 12).

Error-Count

This field shows how many errors have occurred since the last time the *Terminate-Permanent-Error-Window* error report was received. This 2-bit field is incremented each time an error is reported. The counter does not wrap around; if it reaches a value of three, it stops until explicitly cleared by the *Terminate-Permanent-Error-Window* error report. This field identifies an overflow condition for the software system, as shown in Table A-9.

This field defaults to a value of 00_b during RESET.

Table A-9: Interpretation of Error-Count Value

Error Count	Definition
00	No error information
01	A single error report has occurred
10	2 error reports have occurred
11	3 or more error reports have occurred

Permanent-Error Bit

1 = This setting indicates that the error is permanent. Some error types are classified as permanent immediately. Others are classified as permanent if the same error from the same confinement area occurs twice in a row within the permanent error window. This bit is set by the recovery procedures operating in the BXU.

0 = This setting indicates that the error is not permanent.

Error-Record Register (Address 0F0_H)

This register holds the contents of the previous error report to provide additional information to software about the error events in the system. Whenever a new error report message is received the contents of the *Error-Log* register is loaded into the *Error-Record* register. This register should not be written during normal operation because the hardware reads and writes to it.

The contents of this register are not modified during RESET. In a FRC configuration, this register must be written after a *cold* RESET before being read. Otherwise, the master and checker will disagree about the contents of the register since it is not modified during RESET. This register contains information about the state of the system before a *warm* RESET has occurred.

Figure A-10 describes the contents of the register fields and control bits.

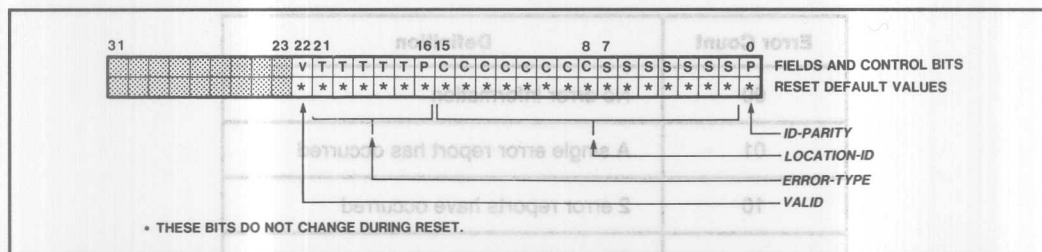


Figure A-10: Error-Record Register Contents

Field and Control Bit Descriptions

ID-Parity Bit This bit is identical to the *ID-Parity* bit in the error report message.

Location-ID This field is identical to the *Confinement-ID* and *Source-ID* fields in the error report message.

Error-Type This field is identical to the error type field in the error report message.

Valid Bit

- 0 = This setting indicates that the other fields in this register are invalid and should be ignored.
- 1 = This setting indicates that the other fields in this register are valid.

FRC Register (Address 064_H)

The contents of this register determine whether a BXU is part of a master/checker pair and how the component responds if it is part of a QMR module. Figure A-11 describes the contents of the register control bits.

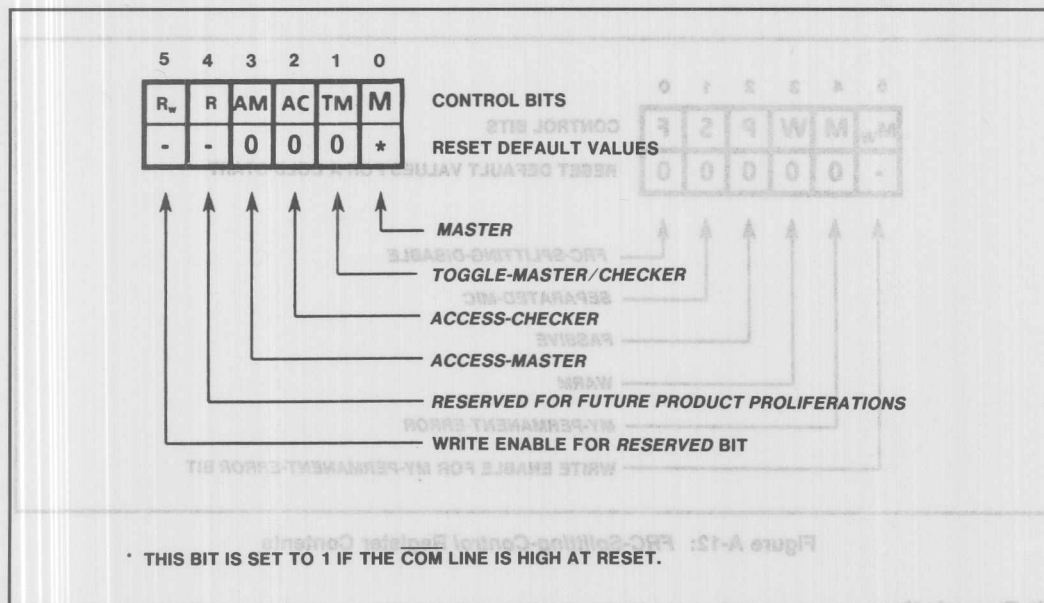


Figure A-11: FRC Register Contents

Bit Descriptions

Master 1 = This setting indicates either that the agent is the Master of a Master/Checker pair, or that there is no Master/Checker pair.

0 = This setting indicates that the agent is the Checker.
If the $\overline{\text{COM}}$ pin is high at RESET, then this bit is set (this part is a master). This read-only bit is not changed by the hardware.

Toggle-Master/Checker 1 = This setting makes the masters and checkers alternate driving the bus on cycle boundaries.

0 = This setting makes the masters always drive the bus.

Access-Checker and Access-Master

See the QMR register for the usage model for these bits.

FRC-Splitting-Control Register (Address 05C_H)

Writing to this register allows a master/checker pair of BXUs to be split into separate functioning components. These register bits are cleared on a *cold* RESET. They are not modified on a *warm* RESET (except as a result of the FRC Splitting algorithm). Figure A-12 describes the contents of the register bits.

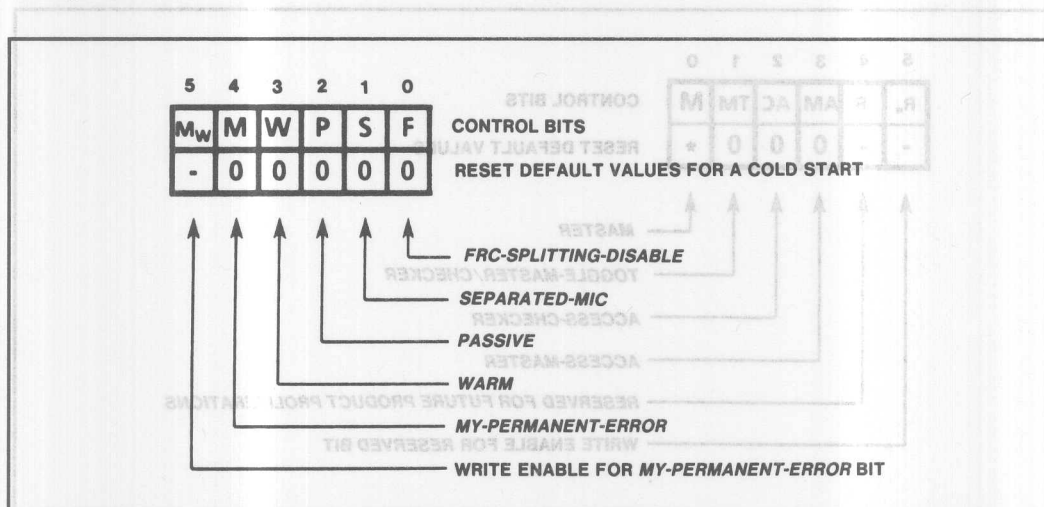


Figure A-12: FRC-Splitting-Control Register Contents

Bit Descriptions

FRC-Splitting-Disable

1 = This setting disables FRC splitting logic.

0 = This setting instructs the BXU to pursue FRC splitting sequence on the next RESET following a permanent error in this module.

Separated-M/C

1 = As a result of FRC splitting or an earlier write to this register, the BXU sets the *Separated-Master/Checker* bit to make the master/checker pair operate as two independent components. Both components act as a master even though one of the components does not have the *Master* bit set. FRC checking on the **BOUT** pin and **MODCHK** pin is disabled. FRC checking remains enabled on all other pins (provides a check on the output buffers).

0 = The BXU clears this bit to allow checking on all pins used for FRC. This master/checker pair does not perform a FRC split.

The purpose of this bit is to allow software to configure a board as either a pair of components or a single master/checker unit. The board layout connects the **BOUT** signal and **MODCHK** signal between the two sets of components.

Passive

This bit is manipulated by the fault tolerance logic in the BXU. See Chapter 13 for details of splitting algorithms that modify this bit.

1 = This setting indicates that the BXU is passive. It cannot drive the AP-bus or the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines. It can track and perform IAC requests sent to its addresses. Normally, a component with *Passive* bit set is the passive component in a FRC split pair of components.

0 = This setting indicates that the component is active and functioning in normal operation.

Warm

1 = The user controls the *Warm* bit. By having software set the *Warm* bit, this bit can be used to indicate that the last RESET was a *warm* RESET. The V_{REF} pin is used to distinguish between a *warm* and *cold* RESET.

0 = This bit is cleared after every *cold* RESET.

My-Permanent-Error

1 = This bit is set if the fault tolerance logic is in the process of setting the *Faulty* bit in the *FT2* register and the *Married* bit in the *QMR* register is not set. This bit is set on permanent module errors when the module is not part of a QMR unit. This bit is an input to the hardware that determines if the module should be split on a *warm* RESET.

0 = This setting indicates that the module has not reported a permanent module error.

Write Enable for the My-Permanent-Error Bit

1 = This setting allows a write operation to the *My-Permanent-Error* bit.

0 = This setting does not allow a write operation to alter the *My-Permanent-Error* bit.

FT1 Register (Address 054_H)

This register in combination with the Fault-Tolerant 2 (FT2) register control the BXU's fault-tolerant capabilities. Figure A-13 describes the contents of the register bits.

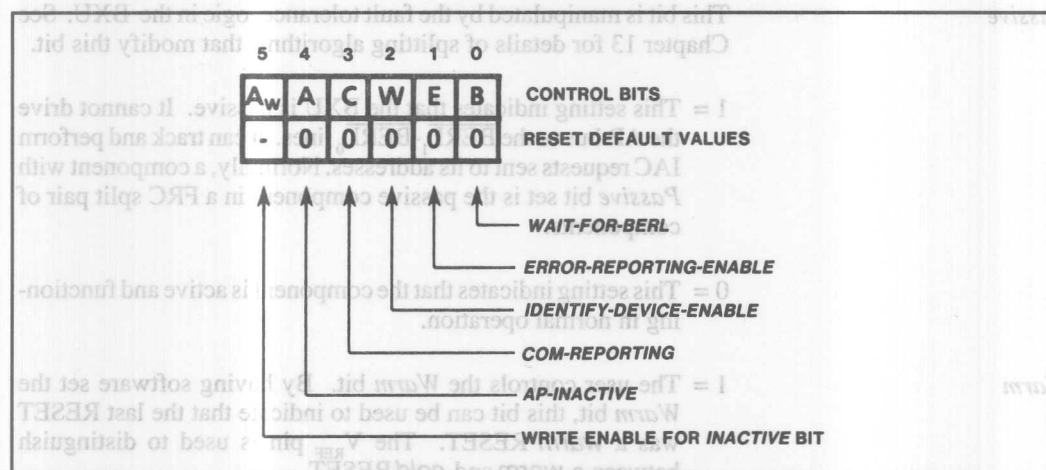


Figure A-13: FT1 Register Contents

Bit Descriptions

Wait-For-BERL

1 = Setting this bit to one commands the BXU to wait until the $\overline{\text{BERL}}_1$ and $\overline{\text{BERL}}_0$ lines can be checked to guarantee valid data from the AP-bus. Data is not used until 0.5 AP-bus clock cycles after the next sampling of data. This action adds 1.5 cycles delay per request.

0 = This setting indicates that data is used without delay.

Error-Reporting-Enable

1 = This setting enables the drivers of the $\overline{\text{BERL}}_1$ and $\overline{\text{BERL}}_0$ pins.

0 = This setting does not allow the BXU to send the error message on the $\overline{\text{BERL}}_1$ - $\overline{\text{BERL}}_0$ lines.

Identify-Device-Disable

1 = This setting ensures proper fault tolerance operation and minimum latency through the BXU for accesses. This bit must be set during normal operation.

0 = This setting enables the "Identify Device" IAC.

This bit must be set to one if fault-tolerant operation is expected.

COM-Reporting

1 = This setting instructs the BXU to send an error report on the error reporting network when the *COM* register is altered using the *COM* register protocol (see Chapter 14). In addition, the *COM-Register-Altered* bit in the *AP-Control* register must not be set.

0 = This setting instructs the BXU not to send an error report when the *COM* register is altered using the *COM* register protocol.

AP-Inactive

1 = This setting disables the AP-bus drivers.

0 = This setting allows the AP-bus drivers to be enabled. This component is functional.

Write Enable for the Inactive Bit

1 = This setting allows a write operation to the *Inactive* bit.

0 = This setting does not allow a write operation to alter the *Inactive* bit.

FT2 Register (Address 0F4_H)

This register holds additional fault-tolerant control parameters that are not contained in the Fault-Tolerant 1 (FT1) register. Figure A-14 describes the contents of the register bits.

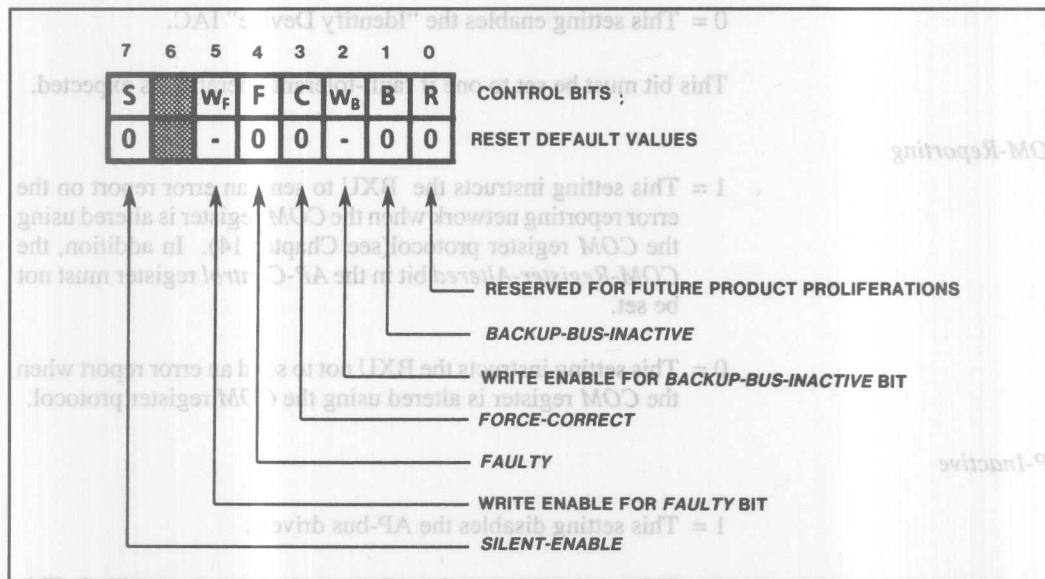


Figure A-14: FT2 Register Contents

Bit Descriptions

Backup-Bus-Inactive

1 = The BXU sets this bit when there is not an active backup AP-bus.

0 = The BXU clears this bit when there is a backup AP-bus for the AP-bus connected to this BXU.

Write Enable for the *Backup-Bus-Inactive* Bit

1 = This setting allows a write operation to the *Backup-Bus-Inactive* bit.

0 = This setting does not allow a write operation to alter the *Backup-Bus-Inactive* bit.

Force-Correct

1 = If the BXU is in Memory mode, then the Correct pin ($\overline{\text{COR}}$) is always asserted.

0 = The $\overline{\text{COR}}$ pin is under the control of the BXU (see Chapter 13 for details on the $\overline{\text{COR}}$ signal).

Faulty

1 = The BXU sets this bit when the module detects a permanent error in its operation and stops operation.

0 = The BXU clears this bit when it is in operation.

Write Enable for the Faulty Bit

1 = This setting allows a write operation to the *Faulty* bit.

0 = This setting does not allow a write operation to alter the *Faulty* bit.

Silent-Enable

This bit is set before the *primary* or *shadow* units of a QMR pair are actually married. When the *Married* bit in the QMR register is set, the *Silent* bit in the QMR register is also set (in both modules), but only one module has the *Silent-Enable* bit set. Thus, only one module is silent.

1 = This setting does not allow the BXU to respond to requests if the *Silent* bit in the QMR register is set.

0 = This setting allows the BXU to respond normally.

Invalidate-Cache Command (Address 15C_H)

A write operation to this register location immediately invalidates all *lines* in the cache directory in the BXU. The data word sent with the write request is ignored. This command can only be executed safely when the cache is disabled.

Faulty
1 = The BXU sets this bit when the module detects a permanent error in its operation and stops operation.

0 = The BXU clears this bit when it is in operation.

Write Enable for the Faulty Bit

1 = This setting allows a write operation to the Faulty bit.

0 = This setting does not allow a write operation to alter the Faulty bit.

Silent-Enable
This bit is set before the primary or shadow bits of a QMR pair are actually married. When the Married bit in the QMR register is set, the Silent bit in the QMR register is also set (in both modules), but only one module has the Silent-Enable bit set. Thus, only one module is silent.

1 = This setting does not allow the BXU to respond to requests if the Silent bit in the QMR register is set.

0 = This setting allows the BXU to respond normally.

LBI-Control Register (Address 148_H)

This is the major control register for the BXU functions on the L-bus. It is used to set the interleaving factor for the cache, to determine whether the BXU should act as an arbitration master on the L-bus, and to determine whether the BXU should function in Memory or Processor mode. Figure A-15 describes the contents of the register fields and bits.

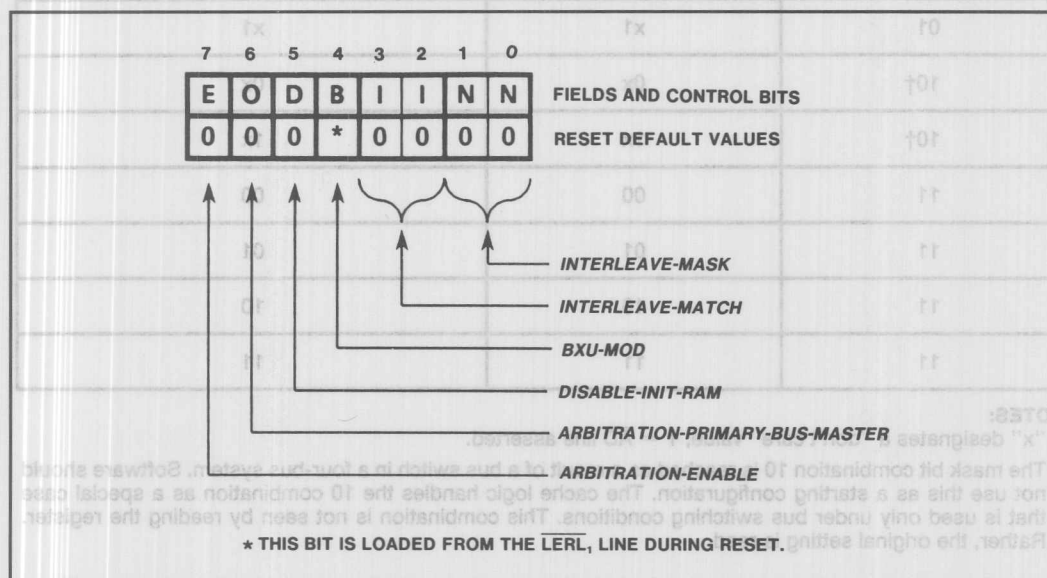


Figure A-15: LBI-Control Register Contents

Before writing to this register, the *Cache-Enable* bit in the *Cache-Configuration* register should be cleared and an *Invalidate-Cache* command sent. This is done because the cache uses the interleaving information in this register to control its operation.

Field and Control Bit Descriptions

Interleave Mask and Match

These two fields determine the interleaving factor and matching for the cache control logic and for the memory address recognizers in the L-bus interface that have interleaving enabled. Table A-10 shows the impact of the different configurations of these bits.

BXU-Mode Bit

This bit selects the mode of operation of the BXU, as shown in Table A-11. This read-only bit is loaded during RESET from the $\overline{\text{LERL}}$ pin. The mode of operation of a BXU can only be set during initialization.

Table A-10: Interleave-Control Bit Settings for Different Configurations

Mask Bits	Match Bits*	AD ₅ , AD ₄ Required for Match*
00	xx	xx
01	x0	x0
01	x1	x1
10†	0x	0x
10†	1x	1x
11	00	00
11	01	01
11	10	10
11	11	11

NOTES:

* "x" designates a "don't care" value; 1 = $\overline{\text{AD}}$ line asserted.

† The mask bit combination 10 is reached as a result of a bus switch in a four-bus system. Software should not use this as a starting configuration. The cache logic handles the 10 combination as a special case that is used only under bus switching conditions. This combination is not seen by reading the register. Rather, the original setting is read.

Table A-11: BXU-Mode Settings

BXU-Mode Bit	Mode	$\overline{\text{LERL}}$, at Reset
0	Memory	Asserted (Low)
1	Processor	Deasserted (High)

Disable-INIT-RAM Bit 0 = This setting enables the INIT-RAM memory recognizer.

1 = This setting disables the INIT-RAM memory recognizer.

Arbitration-PBM/SBM Bit This bit controls the BXU's use of the HOLD/HLDA and HOLDR/HLDAR lines.

1 = The BXU acts as the primary bus master of the L-bus. This provides faster access to the bus when the BXU is operating as the primary bus master.

0 = The BXU acts as a secondary bus master of the L-bus.

Arbitration-Enable Bit

1 = The BXU drives and monitors the arbitration lines in the manner selected by the *Arbitration-PBM/SBM* bit.

0 = The BXU ignores the arbitration lines. If the BXU acts as a primary bus master, it assumes that it has full control of the bus.

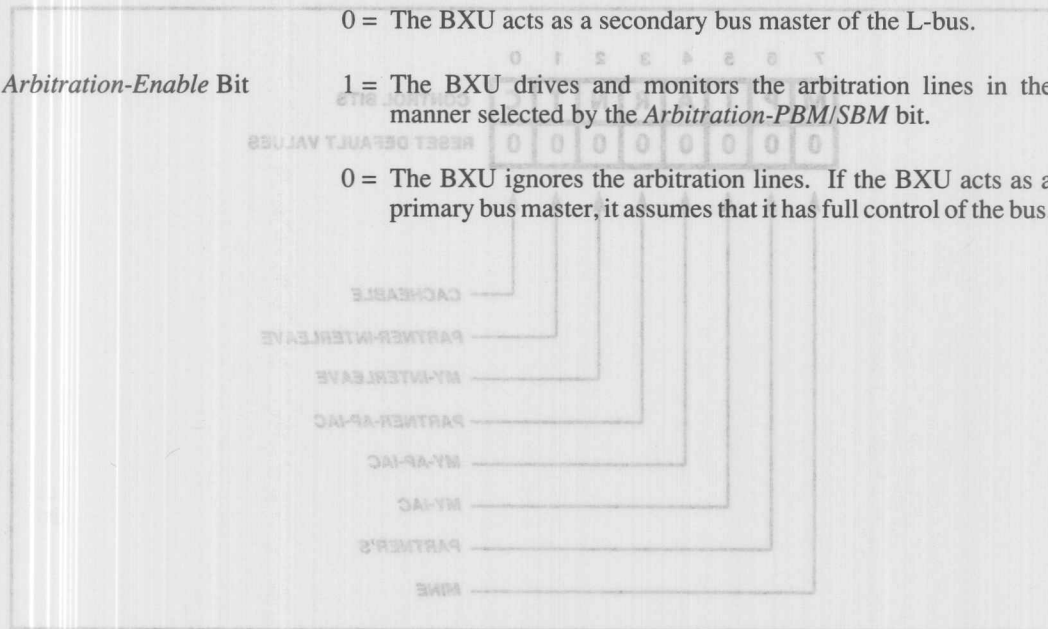


Figure A-18: Local-Bus-Test Register Contents

Bit Descriptions

0 = This setting indicates that the previous request was not cacheable.

1 = This setting indicates that the previous request was cacheable. This means that the CACHE line on the L-bus was asserted. Cache-Inhibit bit was not set in the Match register, the Enable-Cache bit was set in the Cache-Configuration register, and the request was not a RMW or an I/O prefetch.

Local-Bus-Test Register (Address 158_H)

The BXU utilizes the *Local-Bus-Test* register to allow system diagnostics to check on the type of recognition that was done on the previous L-bus request. This register is loaded at the end of each L-bus transaction. A read request to this register provides information about the previous request on the L-bus. Figure A-16 describes the contents of the register fields and control bits.

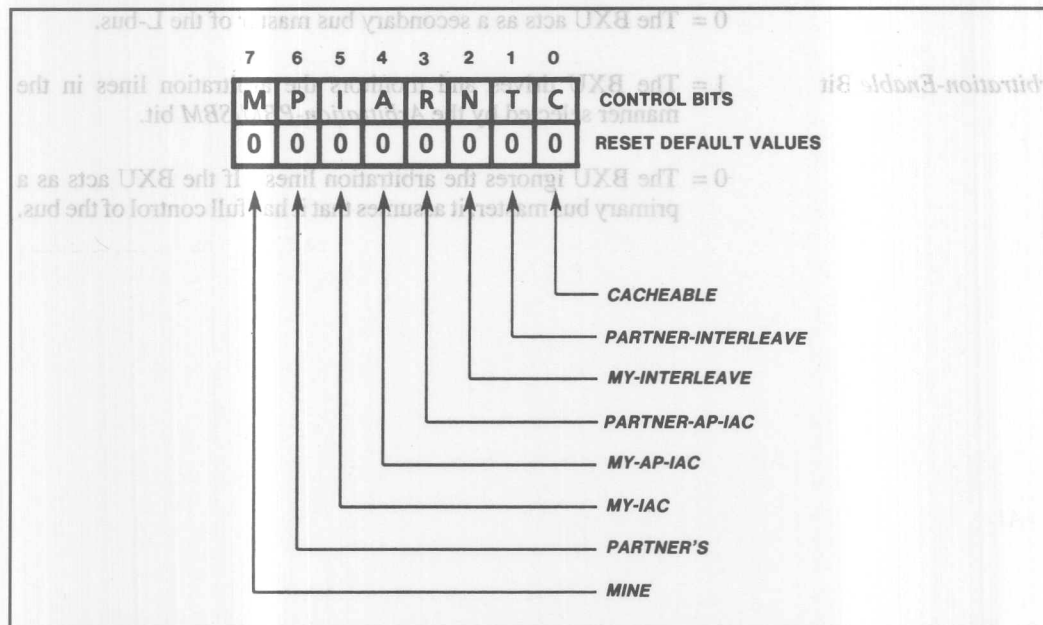


Figure A-16: *Local-Bus-Test* Register Contents

Bit Descriptions

Cacheable

- 0 = This setting indicates that the previous request was not cacheable.
- 1 = This setting indicates that the previous request was cacheable. This means that the CACHE line on the L-bus was asserted, the *Cache-Inhibit* bit was not set in the *Match* register, the *Enable-Cache* bit was set in the *Cache-Configuration* register, and the request was not a RMW or an I/O prefetch.

Partner-Interleave

- 0 = This setting indicates that “my-partner’s” bus was not active or that the previous request did not have AD₅ and AD₄ set such that their value matched the interleaving value for “my-partner’s” bus.

- 1 = This setting indicates that “my-partner’s” bus was active at the time of the request and that the previous request did have AD₅ and AD₄ set such that their value matched the interleaving value for “my-partner’s” bus.

My-Interleave

- 0 = This setting indicates that this bus WAS NOT ACTIVE or that the previous request did not have AD₅ and AD₄ set such that their value matched the interleaving value for this BXU.

- 1 = This setting indicates that this bus WAS ACTIVE at the time of the request and that the previous request did have AD₅ and AD₄ set such that they matched the interleaving value for this BXU.

Partner-AP-IAC

- 0 = The previous request WAS NOT AN IAC that flowed onto the AP-bus of “my-partner’s” BXU.

- 1 = The previous request WAS AN IAC that flowed onto the AP-bus of my-partner BXU.

My-AP-IAC

- 0 = The previous request WAS NOT AN IAC that flowed through this BXU onto the AP-bus.

- 1 = The previous request WAS AN IAC that flowed through this BXU onto the AP-bus.

My-IAC

- 0 = The previous request WAS NOT AN IAC that was handled internally by this BXU.

- 1 = The previous request WAS AN IAC that was handled internally by this BXU.

Partner's

- 0 = The previous request WAS NOT DIRECTED TO this partner's AP-bus.

- 1 = The previous request WAS DIRECTED TO this partner's AP-bus.

Mine

- 0 = The previous request WAS NOT DIRECTED TO this BXU's AP-bus.

- 1 = The previous request WAS DIRECTED TO this BXU's AP-bus.

Lock Register (Address 300_H)

The *Lock* register contains the status of the locks used for RMW operations. Figure A-17 describes the contents of the register fields and control bits.

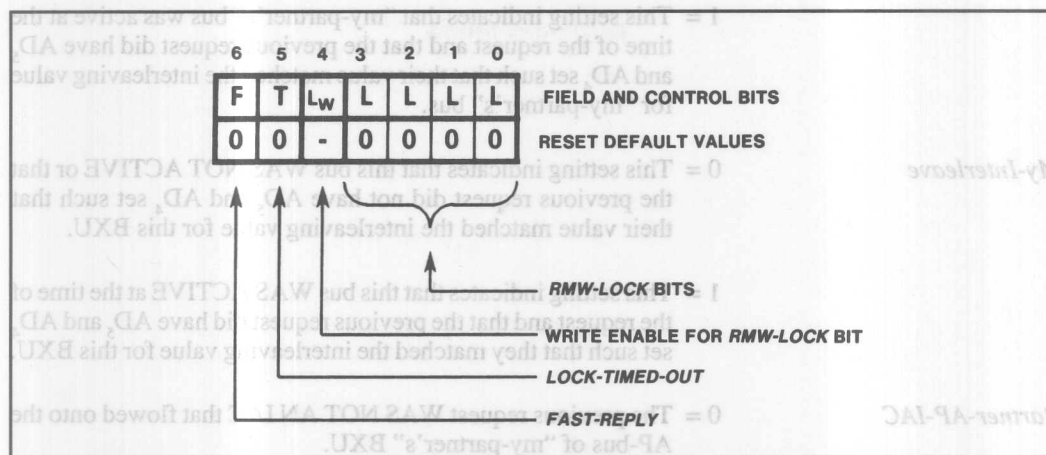


Figure A-17: Lock Register Contents

Field and Control Bit Descriptions

RMW-Lock-Bits

Each of these bits is an RMW Lock for a segment of memory recognized by this BXU. Table A-12 shows the mapping of AD₆ and AD₄ to the *RMW-Lock* bits in this field.

Table A-12: RMW Lock Mapping

RMW-Lock Bit	Address Bit	
	AD ₆	AD ₄
0	0	0
1	0	1
2	1	0
3	1	1

NOTE: 1 = $\overline{\text{AD}}$ line asserted.

A *RMW-Lock* bit is set whenever an incoming RMW-Read request is accepted by the module, and AD₆ and AD₄ match its address segment.

A *RMW-Lock* bit is cleared whenever an incoming RMW-Write request is accepted into the module, and AD_6 and AD_4 match its address segment. If a lock time-out occurs, the *RMW-Lock* bit is also be reset. Each lock times out independently. The duration of the time-out is between 4096 and 8192 clock cycles.

Whenever an error report is received, all *RMW-Lock* bits are automatically set. The timer starts at the beginning of retry (after the end of the transient waiting period). This approach allows the locks to be set and cleared immediately instead of waiting for the request to clear the AP-bus (no possibility of retry). This also provides a way to synchronize the timers for primary-shadow operation.

Write Enable for the *RMW-Lock* Bits

1 = This setting allows a write operation to the *RMW-Lock* bit.

0 = This setting does not allow a write operation to alter the *RMW-Lock* bits.

These bits are protected with a write enable because the BXU hardware modifies these bits as a part of normal operation.

Lock-Timed-Out Bit

1 = This setting indicates that at least one of the four *RMW-Lock* bits was reset by a time-out.

0 = This setting indicates that none of the *RMW-Lock* bits have been reset by a time-out.

These settings for the *RMW-Lock* bits indicate a possible corruption of the integrity of the memory data.

Fast-Reply Bit

1 = The external memory controller is operating with fast memory components. When data begins to flow back to the BXU, a data word is available on every cycle. The BXU does not assert the Reply Deferral signal (\overline{RPYDEF}) on the AP-bus. The data flows directly through the BXU from the L-bus to the AP-bus.

0 = The external memory controller is operating with slow memory components. Data is not delivered on every clock cycle. The BXU asserts \overline{RPYDEF} on the AP-bus. As data flows back from the L-bus, the entire packet is buffered before any data is returned on the AP-bus.

NOTE:

If the Correct signal (\overline{COR}) is asserted (either because the *Force-Correct* bit in the *FT2* register is set, or because the fault tolerance logic asserted it), data is always fully buffered in the BXU, independent of the state of the *Fast-Reply* bit. This allows for the slower timing required by the external ECC circuit when it is performing single bit corrections.

Logical-ID Register (Address 044_H) Logical-ID Register (local) - (Address 144_H)

This register holds the logical identification for the BXU. In every case, all BXUs in the same module share the same logical identification value.

The register address accesses two registers: the *Logical-ID* register located in the AP-bus interface and the *Logical-ID(local)* register located in the L-bus interface. All write operations to register address 044_H (including the Identify Device Order) modify both registers. Read operations to register address 044_H read the register in the AP-bus interface. Read and write operations to register address 144_H use only the *Logical-ID(local)* register in the L-bus interface. Access to the *Logical-ID(local)* register using register address 144_H is primarily made available for test purposes. There is no need to access this register during normal operations. Figure A-18 describes the contents of the register fields.

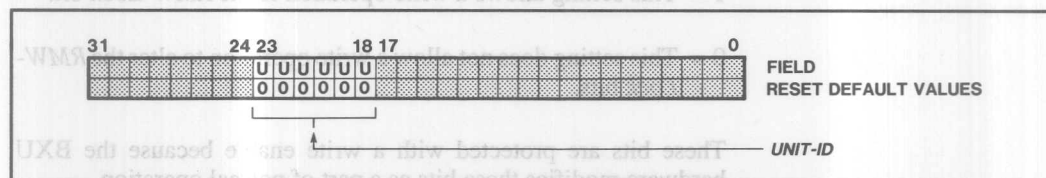


Figure A-18: Logical-ID Register Contents

Field Descriptions

Unit-ID

This field holds the logical identification for the BXU.

Mask and Match Registers (Addresses 160_H through 174_H)

The contents of the *Match* register determine the value of the L-bus address that is recognized by the BXU. This register provides a base address for a partition of memory recognized by the BXU. The contents of the *Mask* register determine whether certain bits should be ignored during the address recognition.

The *Mask* register should always be setup before the *Match* register. This avoids matching on incorrect address patterns.

Before writing to these registers, the *Enable-Cache* bit in the *Cache-Configuration* register should be cleared and an *Invalidate-Cache* command sent. This is performed because the cache uses the information in this register to control its operation.

This description describes the three sets of identical *Mask* and *Match* registers shown in Table A-13.

Table A-13: Address for Match and Mask Registers

Register Address	Register
160 _H	<i>Match</i> ₀
164 _H	<i>Mask</i> ₀
168 _H	<i>Match</i> ₁
16C _H	<i>Mask</i> ₁
170 _H	<i>Match</i> ₂
174 _H	<i>Mask</i> ₂

Figure A-19 describes the contents of the register fields and control bits.

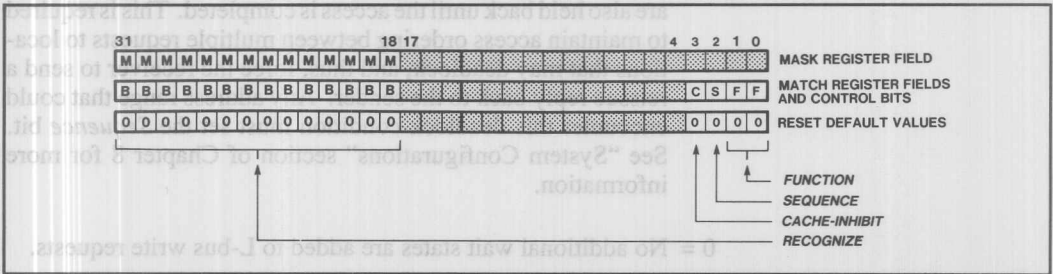


Figure A-19: L-Bus *Mask* and *Match* Register Contents

Function: 00 = This setting disables the *Mask* and *Match* registers. 01 = This setting enables the bus recovery of the *Mask* and *Match* registers. The *Mask* and *Match* registers use the interleave configuration bits in the *LBI-Control* register (which has the option of no interleaving). The cache control logic also uses the interleaving bits to control its operation.

10 = This setting disables interleaving. LAD_5 and LAD_4 are ignored during address matching. If the cache is being used, the *Ignore-Interleave-Control* bit in the *Cache-Configuration* register must be set.

11 = This setting disables the interleaving. LAD_5 and LAD_4 are ignored during address matching. This recognizer is a backup. Requests that use this recognizer are placed in the BXU's internal partner queue to enable tracking of the active bus (i.e., the bus where the BXUs have the *Function* bits set to 10). It only becomes active upon the permanent failure of its partner's bus. If the cache is being used, the *Ignore-Interleave-Control* bit in the *Cache-Configuration* register must be set.

The *Mask* and *Match* registers operation is also conditional on the *Faulty* bit in the *FT2* register and the *Inactive* bit in the *FT1* register. If the *Faulty* bit is set, all of the *Mask* and *Match* registers of the L-bus interface are disabled. If the *Inactive* bit is set in the BXU on this bus, then all requests go to the backup bus. If the *Inactive* bit is set in the BXU on the backup bus, then all requests go to this bus.

Sequence

1 = Any write request that is processed by this recognizer causes the BXU to insert additional wait cycles on the L-bus until the request is completed on the AP-bus. Internal prefetch requests are also held back until the access is completed. This is required to maintain access ordering between multiple requests to locations that may deadlock, and thus, force the receiver to send a reissue reply back to the sender. Any address range that could encounter this deadlock condition must set the *Sequence* bit. See "System Configurations" section of Chapter 8 for more information.

0 = No additional wait states are added to L-bus write requests.

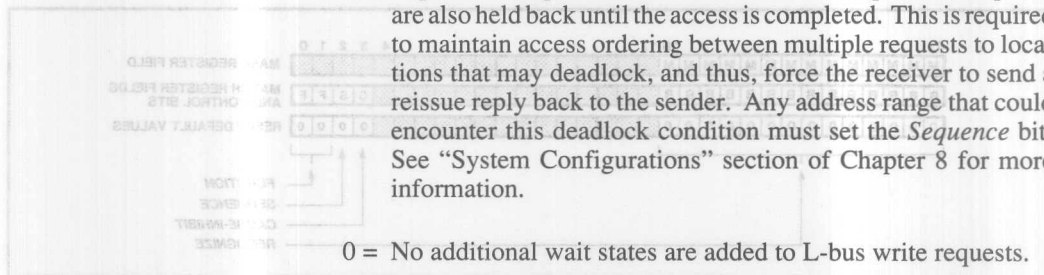


Figure A-19: L-Bus Mask and Match Register Contents

Cache-Inhibit

This bit is for use during cache diagnostics. During normal operation it should always be a zero. For more information on the use of this bit see the "Diagnostic Support Functions" section of Chapter 8.

1 = Requests that are processed by this recognizer are not cached (even if the *Cacheable* bit is asserted). This bit overrides all other cache control bits.

0 = Requests that are processed by this recognizer may go to the cache if the *Function* bits (see below) and other cache control bits are set correctly.

Recognize

In the *Mask* registers, each bit in this field that is set causes the L-bus address bit to be compared against the corresponding *Match* register bit. If a bit is cleared, then that bit position is considered "don't care" during address recognition.

In the *Match* registers, each bit in this field is compared against the corresponding bits in the L-bus address cycle. Thus, these bits provide a base address for the partition of memory that is recognized by this address recognizer.

Maxtime Register (Address 058_H)

The value in this register determines the length of time that the BXUs remain quiescent following the beginning of an error report. Figure A-20 describes the contents of the register bits.

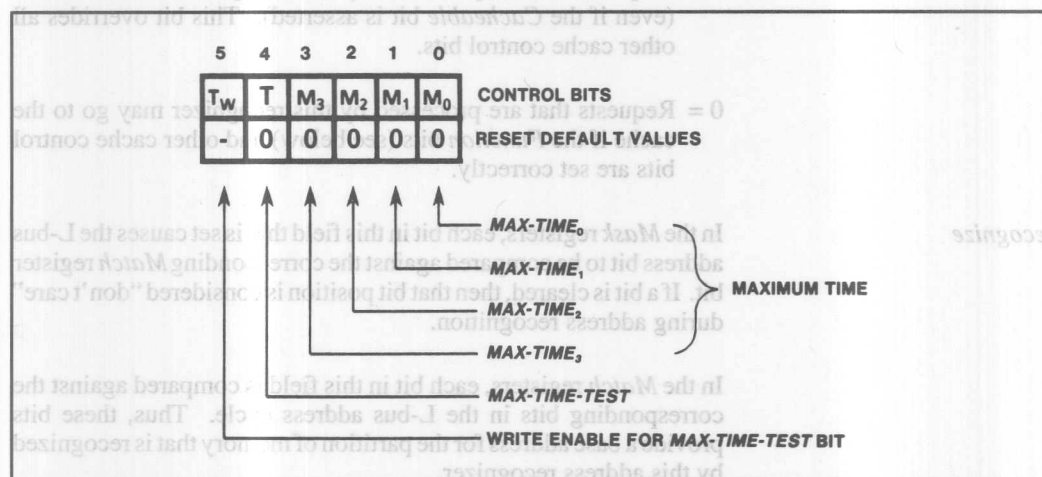


Figure A-20: Maxtime Register Contents

Field and Control Bit Descriptions

Max-Time

These four bits adjust the transient waiting period from 16 microseconds to 0.5 seconds with a 16 MHz Bus Cycle (CLK2 = 32 Mhz). The following formula shows how to increase the time period as a function of the clock frequency:

$$\text{Number of cycles} = 3 + 2^8 + M_0 \cdot 2^{12} + M_1 \cdot 2^{16} + M_2 \cdot 2^{20} + M_3 \cdot 2^{24}$$

Maxtime-Test-Enable

1 = This setting allows testing of the *Maxtime* counter.

0 = This setting inhibits the testing of the *Maxtime* counter (normal setting for system operation).

Write Enables for the Maxtime-Test-Enable Bit

1 = This setting allows a write operation to the *Maxtime-Test-Enable* bit.

0 = This setting does not allow a write operation to alter the *Maxtime-Test-Enable* bit.

Module-Error-ID Register (Address 0E4_H)

This register provides the *Location-ID* field for error reports that originate in a module confinement area. The *Location-ID* field consists of the *Source-ID* and the *Confinement-ID* fields of this register. Figure A-21 describes the data fields and the control bits of this register.

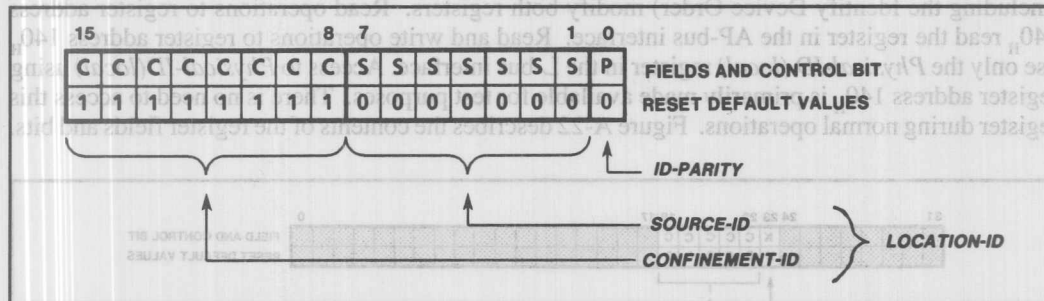


Figure A-21: Module-Error_ID Register Contents

Field and Control Bit Descriptions

ID-Parity Bit

This bit provides even parity for the *Source-ID* and *Confinement-ID* fields in this register. When software loads new values into these fields, it must also load the *ID-Parity* bit to ensure that there is an even number of bits with the value of one in this 16-bit register.

Source-ID

This 7-bit field uniquely identifies a BXU within a confinement area. Software must assign a value to this field. Software uses the *Source-ID* field in the *Error-Log* register to specifically identify which BXU (or Master/Checker pair) had issued the error report message.

Confinement-ID

This 8-bit field uniquely identifies a confinement area within the system. All *Confinement-ID* values in the system must be in the range 4 to 255 (the values of zero to three are reserved for the bus confinement areas). The BXUs use this value in the error report messages to determine which recovery actions to take. Software must assign a value to this field. The *Confinement-ID* field is used when the BXU reports an error in its module confinement area.

Physical-ID Register (Address 040_H) Physical-ID Register(local) - (Address 140_H)

This register contains a unique identifier for a specific BXU on an AP-bus. The register address accesses two registers: the *Physical-ID* register located in the AP-bus interface and the *Physical-ID(local)* register located in the L-bus interface. All write operations to register address 040_H (including the Identify Device Order) modify both registers. Read operations to register address 040_H read the register in the AP-bus interface. Read and write operations to register address 140_H use only the *Physical-ID (local)* register in the L-bus interface. Access to *Physical-ID(local)* using register address 140_H is primarily made available for test purposes. There is no need to access this register during normal operations. Figure A-22 describes the contents of the register fields and bits.

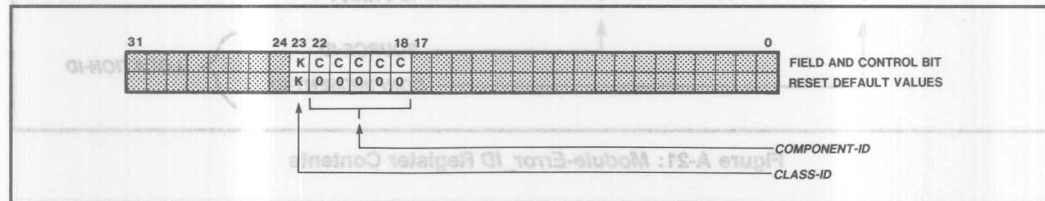


Figure A-22: *Physical-ID* Register Contents

Field and Control Bit Descriptions

Component-ID

The *Component-ID* field allows unique identification of components with the same *Class-ID* field (see following field discussion) on the same bus. The default value of this field is 00_H. This register can be loaded with a unique value by the Identify Device Order, or by performing a write operation.

Class-ID Bit

This bit **must** be set to a value of zero during the initialization routine. The value of one is reserved for future product proliferations.

Prefetch-Control Register (Address 240_H)

The prefetch unit of the BXU uses the *Prefetch-Control* register for control. Figure A-23 describes the contents of the control bits.

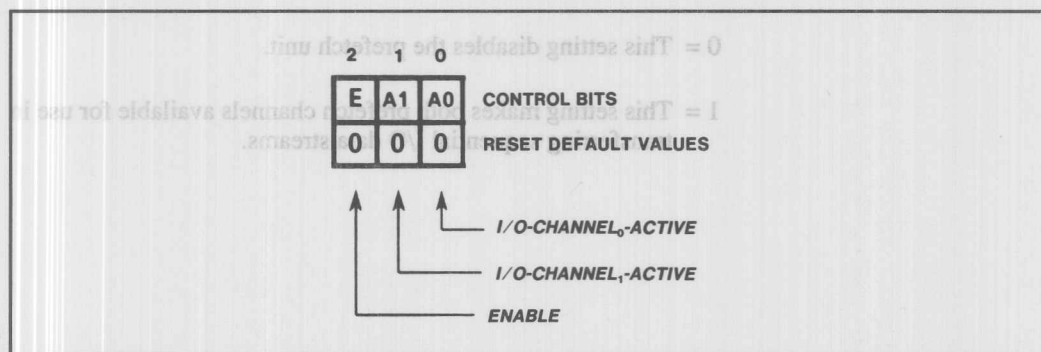


Figure A-23: Prefetch-Control Register Contents

Control Bit Descriptions

I/O-Channel₀-Active

This read-only bit is automatically set whenever a Start-Channel₀ command is performed after the *Enable* bit is set. This bit is automatically cleared whenever a bus switch (attach or detach) occurs that involves this BXU (either its bus or its partner's bus).

1 = This setting indicates that Channel₀ of the prefetch unit is active. The channel actively monitors L-bus traffic for address matches to perform the associated prefetch work.

0 = This setting indicates that Channel₀ is inactive.

I/O-Channel₁-Active

This read-only bit is automatically set whenever a Start-Channel₁ command is performed after the *Enable* bit is set. This bit is automatically cleared whenever a bus switch (attach or detach) occurs that involves this BXU (either its bus or its partner's bus).

1 = This setting indicates that Channel₁ is active. The channel actively monitors the L-bus traffic for address matches to perform the associated prefetch work.

0 = This setting indicates that Channel₁ is inactive.

Enable

The *Enable* bit is automatically cleared whenever a bus switch (attach or detach) occurs that involves this BXU (either its bus or its partner's bus). This means that recovery software needs to explicitly turn the prefetch function back on after the bus switch, if appropriate.

0 = This setting disables the prefetch unit.

1 = This setting makes both prefetch channels available for use in transferring sequential I/O data streams.

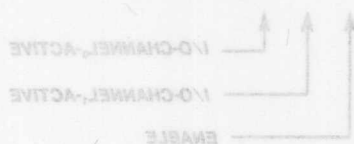


Figure A-23: Prefetch-Control Register Contents

Control Bit Descriptions

This read-only bit is automatically set whenever a Start-Channel command is performed after the *Enable* bit is set. This bit is automatically cleared whenever a bus switch (attach or detach) occurs that involves this BXU (either its bus or its partner's bus).

1 = This setting indicates that Channel₀ of the prefetch unit is active. The channel actively monitors L-bus traffic for address matches to perform the associated prefetch work.

0 = This setting indicates that Channel₀ is inactive.

This read-only bit is automatically set whenever a Start-Channel command is performed after the *Enable* bit is set. This bit is automatically cleared whenever a bus switch (attach or detach) occurs that involves this BXU (either its bus or its partner's bus).

1 = This setting indicates that Channel₁ is active. The channel actively monitors the L-bus traffic for address matches to perform the associated prefetch work.

0 = This setting indicates that Channel₁ is inactive.

Primary-Catastrophe Command (Address 108_H)

A write operation to this register location causes the BXU to generate a *Primary-Catastrophe* error report on the serial error reporting network. The data word sent with the write request is ignored.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with the value of one in the *COM* register.

Figure A-24 describes the contents of the register fields and control bits.

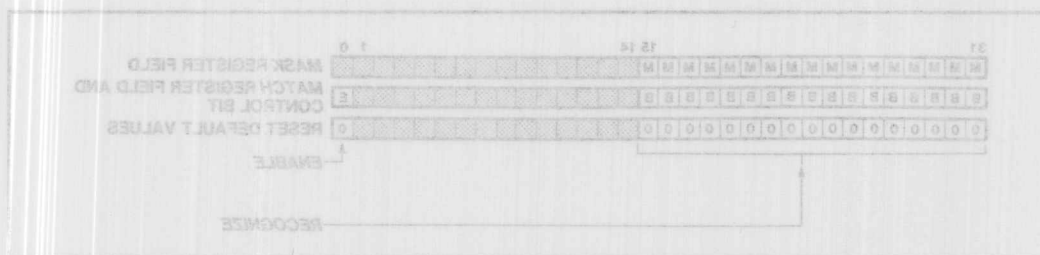


Figure A-24: Private Memory Mask and Match Register Contents

Field and Control Bit Descriptions

Recognize

In the Mask registers, each bit in this field that is set causes the corresponding I-bus address bit to be compared against the corresponding Match register bit. If a bit is cleared, then that bit position is considered a "don't care" during address recognition.

In the Match registers, each bit in this field is compared against the corresponding bits in I-bus address cycles. Thus, these bits provide a base address for the partition of memory that is recognized by this address recognizer.

Enable

0 = This setting disables the Mask and Match registers.

1 = This setting enables the Mask and Match registers. The operation of the Mask and Match registers is also conditional on the Faulty bit in the FTY register (module failure). If the Faulty bit is set, then the private memory Mask and Match registers are disabled.

Private Memory *Mask* and *Match* Registers (Add. 178_H, 17C_H)

The private memory *Mask* and *Match* registers provide the ability to support SRAM on the L-bus as a normal memory, instead of cache. **The *Mask* register should always be setup before the *Match* register.** This is required to avoid matching on incorrect address patterns.

When a match occurs on the private memory recognizer, the \overline{WY}_1 pin is asserted and the address bit from the LAD_{14} pin is driven on the \overline{WY}_0 pin. If the address range of the private memory recognizer overlaps the range of an AP-bus recognizer (160_H-174_H), then the request is considered to be to the private memory.

Figure A-24 describes the contents of the register fields and control bits.

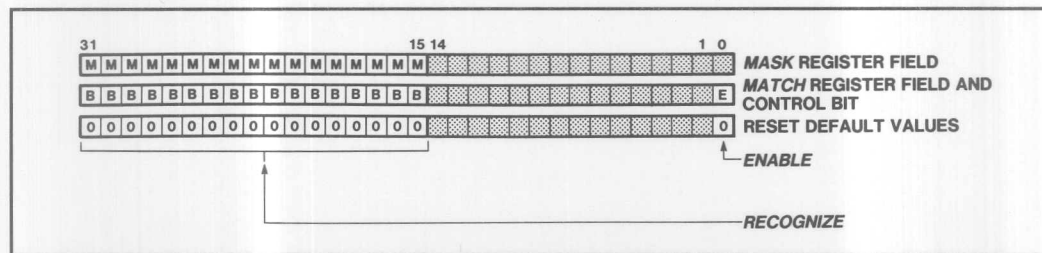


Figure A-24: Private Memory *Mask* and *Match* Register Contents

Field and Control Bit Descriptions

Recognize

In the *Mask* registers, each bit in this field that is set causes the corresponding L-bus address bit to be compared against the corresponding *Match* register bit. If a bit is cleared, then that bit position is considered a “don’t care” during address recognition.

In the *Match* registers, each bit in this field is compared against the corresponding bits in L-bus address cycles. Thus, these bits provide a base address for the partition of memory that is recognized by this address recognizer.

Enable

0 = This setting disables the *Mask* and *Match* registers.

1 = This setting enables the *Mask* and *Match* registers. The operation of the *Mask* and *Match* registers is also conditional on the *Faulty* bit in the *FT2* register (module failure). If the *Faulty* bit is set, then the private memory *Mask* and *Match* registers are disabled.

Processor-Priority Registers (Address 000_H and 020_H)

The register with address 000_H (Processor-Priority₀) holds the priority of the task (process) which processor₀ on the BXU's L-bus is currently executing. Similarly, the register with address 020_H (Processor-Priority₁) holds the priority of the task (process) which processor₁ on the BXU's L-bus is currently executing. Figure A-25 describes the contents of the register fields and bits.

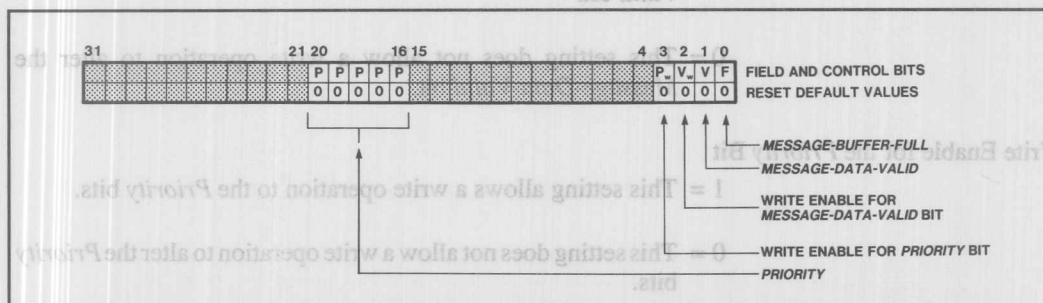


Figure A-25: Processor-Priority Register Contents

Field and Control Bit Descriptions

Message-Buffer-Full Bit

This read-only bit shows the state of the $\overline{\text{IAC}}$ pin for processor₀ or processor₁. The state of the $\overline{\text{IAC}}$ pin may be changed by changing the value of the *Message-Data-Valid* bit.

1 = This setting indicates that an IAC message is waiting for the 80960MC processor. Any subsequent IAC message requests received for this processor are sent the No Acknowledgement reply. The $\overline{\text{IAC}}$ pin is asserted (low).

0 = This setting indicates that the IAC buffer is empty. The $\overline{\text{IAC}}$ pin is not asserted (high).

Message-Data-Valid Bit

1 = This setting indicates that the BXU, which set the *Message-Buffer-Full* bit, has the IAC message data. $\overline{\text{IAC}}$ pin for processor₀ or processor₁ is asserted. This bit is only set after the reply to the message is sent on the AP-bus. When the *Message-Data-Valid* bit is set, the *Message-Buffer-Full* bit is also set. The *Message-Buffer-Full* bit is set in all BXUs in the module, but the *Message-Data-Valid* bit is only set in a single BXU.

0 = This setting indicates that the message buffer does not hold valid data.

Priority

These five bits show the priority of the activity that the processor is currently executing. Priority 0 is the lowest priority and 31 is the highest.

Write Enable for Message-Data-Valid Bit

1 = This setting allows a write operation to the Message-Data-Valid bit.

0 = This setting does not allow a write operation to alter the Message-Data-Valid bit.

Write Enable for the Priority Bit

1 = This setting allows a write operation to the Priority bits.

0 = This setting does not allow a write operation to alter the Priority bits.

Figure A-25: Processor-Priority Register Contents

Field and Control Bit Descriptions

This read-only bit shows the state of the IAC pin for processor or processor. The state of the IAC pin may be changed by changing the value of the Message-Data-Valid bit.

Message-Buffer-Full Bit

1 = This setting indicates that an IAC message is waiting for the 80960MC processor. Any subsequent IAC message requests received for this processor are sent to the Acknowledgment reply. The IAC pin is asserted (low).

0 = This setting indicates that the IAC buffer is empty. The IAC pin is not asserted (high).

1 = This setting indicates that the BXU, which set the Message-Buffer-Full bit, has the IAC message data. IAC pin for processor or processor, is asserted. This bit is only set after the reply to the message is sent on the AP-bus. When the Message-Data-Valid bit is set, the Message-Buffer-Full bit is also set. The Message-Buffer-Full bit is set in all BXUs in the module, but the Message-Data-Valid bit is only set in a single BXU.

Message-Data-Valid Bit

0 = This setting indicates that the message buffer does not hold valid data.

QMR Register (Address 0E0_H)

The contents of this register determine whether a module is part of a QMR pair, and whether it should function as the *primary* or *shadow* unit. Figure A-26 describes the contents of the register fields.

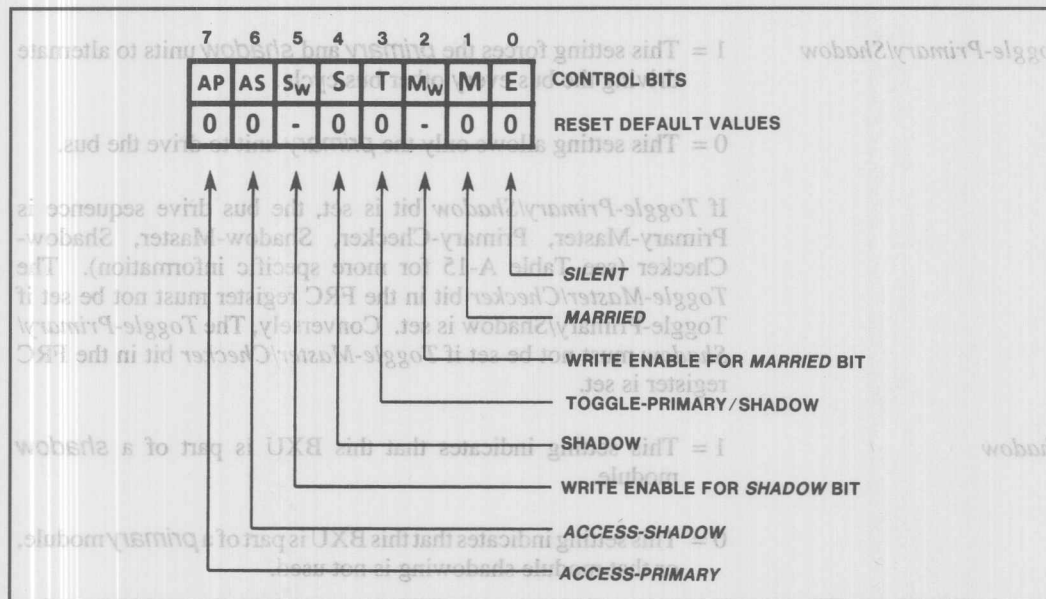


Figure A-26: QMR Register Contents

Control Bit Descriptions

Silent

1 = This setting does not allow the BXU to reply to bus requests if the *Silent-Enable* bit in the *FT2* register is set. In other words, the *Silent* bit selects which module does not drive the AP-bus. This bit is automatically set by a *Sync-Refresh* command that may be issued to this BXU or its partner. When the *Silent* bit is set, the *Toggle-Primary/Shadow* bit must be cleared. The *Silent* bit has the same value in both the *primary* and *shadow* units. The *Silent* bit is also used to control the precise time at which silent operation begins.

0 = Normal operation.

Married

1 = This setting indicates that the FRC pair is married to another FRC pair.

0 = This setting indicates that the FRC pair is not married.

Write Enable for the *Married* bit

1 = This setting allows a write operation to the *Married* bit.

0 = This setting does not allow a write operation to alter the *Married* bit.

Toggle-Primary/Shadow

1 = This setting forces the *primary* and *shadow* units to alternate driving the bus every other bus cycle.

0 = This setting allows only the *primary* unit to drive the bus.

If *Toggle-Primary/Shadow* bit is set, the bus drive sequence is Primary-Master, Primary-Checker, Shadow-Master, Shadow-Checker (see Table A-15 for more specific information). The *Toggle-Master/Checker* bit in the FRC register must not be set if *Toggle-Primary/Shadow* is set. Conversely, The *Toggle-Primary/Shadow* must not be set if *Toggle-Master/Checker* bit in the FRC register is set.

Shadow

1 = This setting indicates that this BXU is part of a *shadow* module.

0 = This setting indicates that this BXU is part of a *primary* module, or that module shadowing is not used.

This information determines whether this BXU drives the AP-bus and responds to an IAC request that has the *Access* bit set in the IAC address.

Write Enable for the *Shadow* Bit

1 = This setting allows a write operation to the *Shadow* bit.

0 = This setting does not allow a write operation to alter the *Shadow* bit.

Access-Shadow and *Access-Primary*

The value of these bits determines which BXU in a QMR configuration responds to an IAC request using a logical address (IAC access type 0010_B) when the *Access* bit is set. In addition, this bit determines which BXU drives the AP-bus in the QMR module.

The *Married*, the *Shadow*, *Access-Shadow*, the *Access-Primary*, the *Access-Master* and *Access-Checker* bits are used to determine which BXU responds to an IAC request type 0010_B when the *Access* bit is set, as shown in Table A-14. Note that requests using the

Access bit must originate from the AP-bus. The registers that are not part of the AP-bus interface logic ignore the *Access* bit, and all BXUs where the appropriate portion of the address matches the *Unit-ID* field in the *Logical-ID* register perform the request.

Table A-14: Determination of Which BXU Performs an IAC Type 0010₈

Configuration	BXU* that Performs Request	Conditions					
		Access-Primary Bit in QMR Register	Access-Shadow Bit in QMR Register	Access-Master Bit in FRC Register	Access-Checker Bit in FRC Register	Married Bit in QMR Register	Shadow Bit in QMR Register
QMR	Shadow Checker	—	1	—	1	1	1
	Shadow Master	—	1	1	—	1	1
	Primary Checker	1	—	—	1	1	0
	Primary Master	1	—	1	—	1	0
FRC	Checker	1	0	0	1	0	0
	Master	1	0	1	0	0	0

NOTE:

* The BXU is identified by the *Master* bit in the *FRC* register and the *Shadow* bit in the *QMR* register. For example, if the *Shadow* bit and the *Master* bit are set, the BXU is a Shadow Master.

The determination of which BXU replies to the requests in a QMR or FRC configuration is listed in Table A-15. No other bits determine which BXU replies. Note that Table A-15 applies to all requests. This means that IAC requests using a physical address are not allowed while toggling is enabled. Even though multiple components may perform the request, only one component replies on the bus.

Configuration	BXU ¹ that Replies	Conditions			
		Silent Bit in QMR Register	Married Bit in QMR Register	Toggle-Primary/Shadow Bit in QMR Register	Toggle-Master/Checker Bit in FRC Register
FRC	Master (M)	0	0	—	0
	Toggle M, C, M, C, etc.	0	0	0	1
QMR	Primary Master (PM)	0	1	0	0
	Toggle PM, PC ² , PM, etc.	0	1	0	1
	Toggle PM, PC, SM ³ , SC, PM, etc.	0	1	1	0
All	No BXU Replies	1 ⁴	—	0	—

NOTES:

1. The BXU is identified by the *Master* bit in the *FRC* register and the *Shadow* bit in the *QMR* register. For example, if the *Shadow* bit and the *Master* bit are set, the BXU is a shadow master.
2. PC is an abbreviation for Primary Checker.
3. SM is an abbreviation for Shadow Master; SC is an abbreviation for Shadow Checker.
4. The *Silent-Enable* bit in the *FT2* register is also set.

Shadow-Catastrophe Command (Address 10C_H)

A write operation to this register location causes the BXU to generate a *Shadow-Catastrophe* error report on the serial error reporting network. The data word sent with the write request is ignored.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with the value of one in the *COM* register.

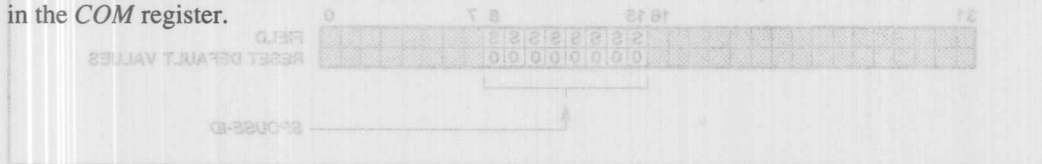


Figure A-27: Spouse-ID Register Contents

Field Descriptions

This 8-bit field corresponds to the *Confinement-ID* field in the *Module-ID* register of the BXUs to which the module is married.

Spouse-ID

Spouse-ID Register (Address 0C4_H)

This register is used by BXUs in a QMR module configuration. For a married *Primary/Shadow* pair, this register holds the identification information of its “mate”. Figure A-27 describes the contents of the register fields.

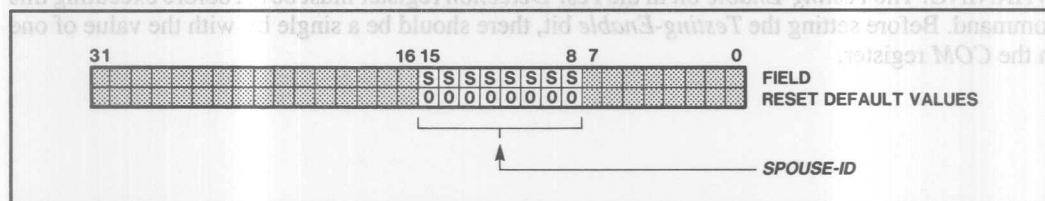


Figure A-27: Spouse-ID Register Contents

Field Descriptions

Spouse-ID

This 8-bit field corresponds to the *Confinement-ID* field in the *Module-ID* register of the BXUs to which the module is married.

Synchronize-Refresh Command (Address 11C_H)

A write operation to this register location address causes the BXU to generate the *Sync-Refresh* error report on the serial error reporting network. The data word sent with the write request is ignored. The BXUs in Memory mode receiving the *Sync-Refresh* error report perform the following actions:

- Assert the Force Refresh (FRF) pin for one cycle at the beginning of the transient waiting period.
- Set the *Enable* bit in the *AP-Match* register to turn on AP-bus address recognizer. This is done by means of an error report because it is the only time that the memory module can be guaranteed to be idle.
- Set the *Silent* bit in the *QMR* register. This action causes the module that previously had the *Silent-Enable* bit set in the *FT2* register to become silent (it will not reply to any accesses).

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with the value of one in the *COM* register.

*** THIS BIT IS SET TO A VALUE OF ONE IF MODCHK₀ IS ASSERTED,
OTHERWISE IT IS SET TO ZERO.**

**† THIS BIT IS SET TO A VALUE OF ONE IF BOUT₀ IS ASSERTED,
OTHERWISE IT IS SET TO A VALUE OF ZERO.**

Field Description

These two bits uniquely identify the BXU as attached to one of up to four buses. At RESET, if $\overline{\text{MODCHK}}$ and $\overline{\text{BOUT}}$ are pulled high (not asserted) by external resistors, the *AP-Bus-ID* field is set at a value of 00.

Terminate-Permanent-Error-Window Command (Address 110_H)

A write operation to this register causes the BXU to generate a *Terminate-Permanent-Error-Window* error report on the serial error reporting network. The data word sent with the write request is ignored. This action closes the permanent error window, so that a reoccurrence of a previous error is not recorded as permanent.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with a value of one in the *COM* register.

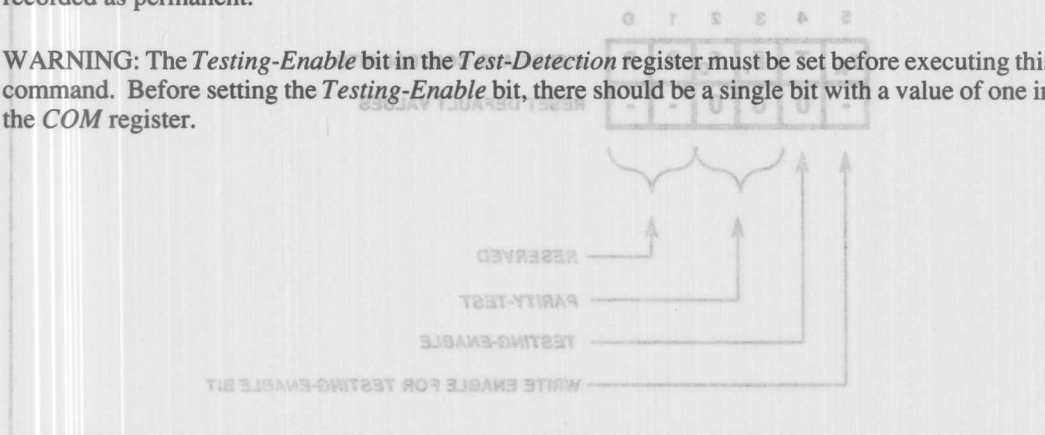


Figure A-29: Test-Detection Register Contents

Field and Control Bit Descriptions

1 = This setting instructs the BXU to test the parity checking logic. Before setting this bit, there must be a single value of one in the *COM* register. This must be done to prevent a FRC error from occurring as a result of an incorrect FRC test. (The FRC test occurs automatically as soon as the *Testing-Enable* bit is set.)

0 = This setting causes the BXU not to test the parity checking logic.

Normal operation checks that the parity tree is working correctly. If there was a failure in the parity tree, then the BXU signals a parity error because its computed parity differs from that generated by the sending component. However, normal operation does not check the logic that compares the check bit with the result computed from the internal parity tree. Moreover, neither the inputs nor the output are checked during normal operation because there normally is parity agreement. The parity testing bits allow testing of this logic.

Parity-Test

Test-Detection Register (Address 060_H)

Figure A-29 describes the contents of the register fields and bits

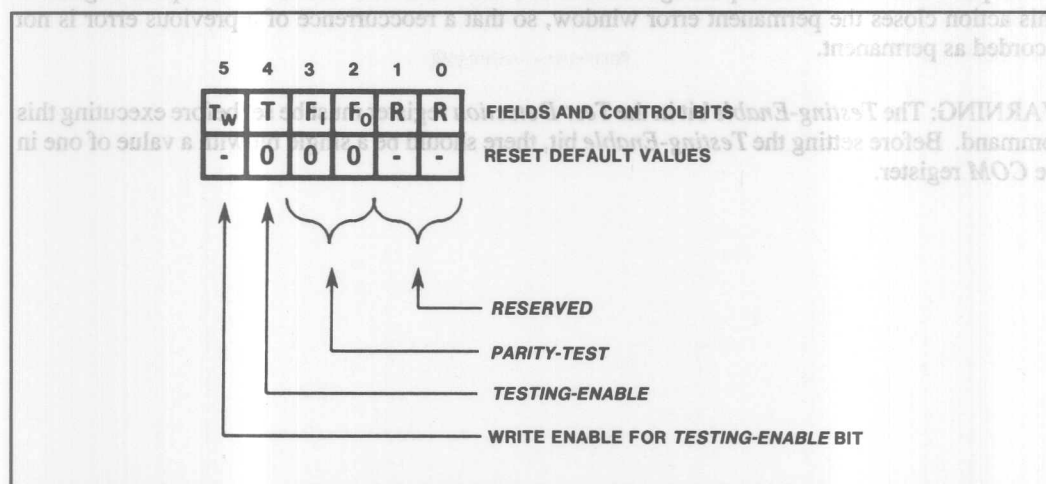


Figure A-29: Test-Detection Register Contents

Field and Control Bit Descriptions

Parity-Test

- 1 = This setting instructs the BXU to test the parity checking logic. Before setting this bit, there must be a single value of one in the *COM* register. This must be done to prevent a FRC error from occurring as a result of an incorrect FRC test. (The FRC test occurs automatically as soon as the *Testing-Enable* bit is set.)
- 0 = This setting causes the BXU not to test the parity checking logic.

Normal operation checks that the parity tree is working correctly. If there was a failure in the parity tree, then the BXU signals a parity error because its computed parity differs from that generated by the sending component. However, normal operation does not check the logic that compares the check bit with the result computed from the internal parity tree. Moreover, neither the inputs nor the output are checked during normal operation because there normally is parity agreement. The parity testing bits allow testing of this logic.

The *Parity-Test* bits invert the $\overline{\text{CHK}}$ pin(s) sampled from the bus. This action creates a parity error. The parity comparing logic is expected to detect the error. The component reports an error if the *Parity-Test* bit is set and a parity error is detected. For a particular parity tree to be tested, the appropriate *Parity-Test* bit must be set. Table A-16 shows the mapping of *Parity-Test* bit to which part of the tree is corrupted.

Parity testing can be done on-line, but a real parity error (i.e., non-forced) during the test may go undetected. The success of the test is noted by checking the *Error-Log* register after the test.

Table A-16: *Parity-Test* Bits Versus Parity Tree Corrupted

Parity Test Bits		Function
F1	F0	
x	1	$\overline{\text{CHK}}_0$ parity error is forced
1	x	$\overline{\text{CHK}}_1$ parity error is forced
1	1	Illegal

Testing-Enable Bit

1 = This setting enables the BXU to execute the testing commands.

0 = This setting disables the BXU from executing the testing commands.

This bit indicates a test function is enabled. When this bit has a value of one, then FRC testing is automatically enabled. For correct operation, the *COM* register must have a single bit with a value of one while the *Testing-Enable* bit is set. This bit is reset by any error report. The resetting of this bit by the error report at the end of a test ensures that the test is not retried in the event that another error report is generated as a result of the test. This eliminates the possibility of any infinite loops. This bit is also cleared when the *COM* register is altered using the *COM* register protocol (see Chapter 14 for details on the protocol). This bit defaults to a value of zero.

Write Enable for the Testing-Enable Bit

1 = This setting allows a write operation to the *Testing-Enable* bit. Write protection is provided because access to this register is retried when the test results in an error report.

0 = This setting does not allow a write operation to alter the *Testing-Enable* bit.

Test-Report Command (Address 104_H)

The *Test-Report* command forces the BXU to test the error reporting network. The type of error report generated is determined by the contents of the *Test-Type* register. A write operation to this register location causes the BXU to execute a *Test-Report* command. The data word sent with the write request is ignored.

WARNING: The *Testing-Enable* bit in the *Test-Detection* register must be set before executing this command. Before setting the *Testing-Enable* bit, there should be a single bit with a value of one in the *COM* register.

Table A-16: Parity-Test Bits Versus Parity Tree Corruption

Function	Parity Test Bits	
	P0	P1
CHK, parity error is forced	1	x
CHK, parity error is forced	x	1
Illegal	1	1

1 = This setting enables the BXU to execute the testing commands.
0 = This setting disables the BXU from executing the testing commands.

This bit indicates a test function is enabled. When this bit has a value of one, then PRC testing is automatically enabled. For correct operation, the COM register must have a single bit with a value of one while the *Testing-Enable* bit is set. The bit is reset by any error report. The resetting of this bit by the error report at the end of a test ensures that the test is not retried in the event of another error report. This eliminates the possibility of any infinite loops. This bit is also cleared when the COM register is altered using the COM register protocol (see Chapter 14 for details on the protocol). This bit defaults to a value of zero.

Write Enable for the Testing-Enable Bit
1 = This setting allows a write operation to the *Testing-Enable* bit. Write protection is provided because access to this register is retried when the test results in an error report.
0 = This setting does not allow a write operation to alter the *Testing-Enable* bit.

Test-Type Register (Address 0C0_H)

The *Test-Type* register is used in conjunction with the *Test-Report* command. The *Test-Report* command forces the BXU to test the error reporting network. The *Test-Type* register determines the type of error report that is generated. Figure A-30 describes the contents of the register field.

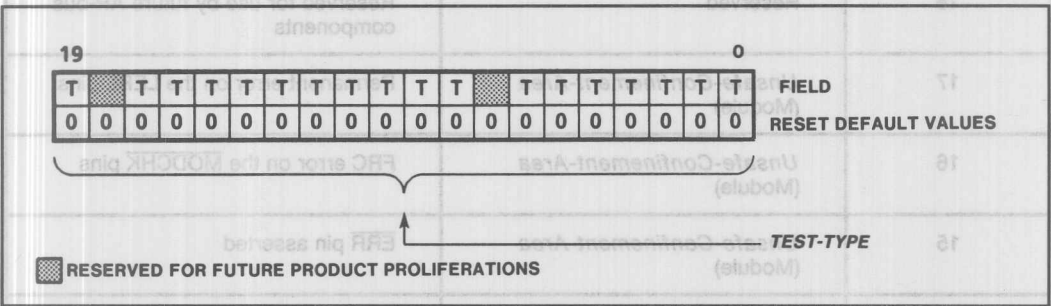


Figure A-30: Test-Type Register

Field Descriptions

Test-Type

This 20-bit field determines the type of error that is generated. Each test bit in this register specifies a specific error type, as shown in Table A-17. Multiple errors may be loaded to test the priority circuits of the BXU. Note that the contents of this field controls the test types generated. The actual test occurs as a result of a *Test-Report* command. The specified tests are performed if the *Testing-Enable* bit in the *Test-Detection* register is set.

If all the bits in the *Test-Type* field are set to a value of zero and a *Test-Report* command is issued, then the result of the command is no operation and no error report is generated. This action prevents sending another error report when the *Test-Report* command is retried at the end of the reporting and recovery cycle. If the *Test-Detection* register is not enabled and the *Test-Report* command is issued, then the result of the command is no operation and no error report is generated. This action also prevents sending another error report because the error report causes the *Test-Detection* register to reset the *Testing-Enable* bit in all BXUs.

NOTE

In order to insure software compatibility with members of the 960 product family RESERVED error condition bits must remain at their reset default values of zero.

Table A-17: Settings for Test-Type Field

Bit Number	Type of Error	Comment
19	Unsafe-Confinement-Area (Bus)	Permanent error on the BERL pins
18	Reserved	Reserved for use by future AP-bus components
17	Unsafe-Confinement-Area (Module)	Permanent error on the LERL pins
16	Unsafe-Confinement-Area (Module)	FRC error on the MODCHK pins
15	Unsafe-Confinement-Area (Module)	ERR pin asserted
14	Primary-Catastrophe	
13	Shadow-Catastrophe	
12	Error-Reporting-Error (Bus)	Transient error on the BERL pins
11	Error-Reporting-Error (Module)	Transient error on the LERL pins
10	Bus-Arbitration	
9	Bus-Parity	
8	Component	
7	Reserved	Reserved for use by future AP-bus components
6	Uncorrectable-Array-Error	
5	Correctable-ECC	
4	Com-Altered	
3	Attach-Bus	
2	Detach-Bus	
1	Terminate-Permanent-Error-Window	
0	Sync-Refresh	

NOTE: In order to ensure software compatibility with future members of the 960 product family reserved error condition bits must remain at their reset default values of zero.

Index

INDEX

memory address recognition	8-12
AP-Control Register	8-12
definition	8-12
82586/80960MC Interface	5-7
M8259A/80960MC Interface	5-5
82786/80960MC Interface	5-12
used in an error type	8-12
AP-Max Register	8-12
Access Bit	8-12, 12-16
Active Module	8-43
Active/Passive Module	8-14
AD ₃₁ -AD ₀ (Address/Data) Lines on the AP-Bus	7-3
definition	7-3
used in FRC	11-4
used with request and reply packets	7-11
Address Block for Cache	8-14
Address Decoder	5-3
I/O interface	4-2
memory interface	4-2
Address Latch/Demultiplexer	5-4
I/O interface	4-2
memory interface	4-2
ADS (Address Status) Signal	3-4
definition	3-4
timing diagram	3-9
used by the 82786 interface	5-13
used by the burst logic	4-3
used by the timing control logic	5-3
ALE (Address Latch Enable) Signal	3-4
definition	3-8
timing diagram	3-8
used by an address latch/demultiplexer 4-2, 5-3	4-7
used by the byte enable latch	4-8
used by the SRAM interface logic	4-9
used in the SRAM interface logic	4-9
AP-Bus	13-22
communication between buses	11-1
confinement area	7-1
definition	7-24
protocol	13-18
recovery	7-3
signal groups	7-32
signal timing	11-2
time-outs	7-2
topology	8-8
AP-Bus Interface Logic on the BXU	8-4
bus interleaving	14-5
functions, list of	8-26
IAC address recognition (Identify Device Order)	8-29, 8-30
IAC address recognition (Logical)	8-28
IAC address recognition (Message)	8-28
IAC address recognition (Physical)	8-28

memory address recognition	8-4
AP-Control Register	
definition	A-8
used during initialization	14-5
used during the parameterization phase of initialization	14-8
used for bus switching	13-17
used in an error type	12-13
AP-Mask Register	
definition	A-10
example of use during identification phase of initialization	14-24, 14-27
interleaving	8-8
used for address recognition on the AP-bus	8-4
used for marrying modules by a special agent	14-12
used for processor module recovery	13-10
used to marry processor modules using another module	13-11
AP-Match Register	
definition	A-10
example of use during identification phase of initialization	14-24, 14-27
interleaving	8-8
used for address recognition on the AP-bus	8-4
used for marrying modules by a special agent	14-11
used to marry processor modules using another module	13-10
ARB₃-ARB₀ (Arbitration) Signals	
definition	7-4
signal duplication in fault-tolerant systems	11-2
timing	7-32
used in FRC	11-5
Arbitration	
AP-bus example	7-26
L-bus example	3-18
protocol for the AP-bus	7-24
protocol for the L-bus	3-17
timing on the L-bus	3-19
Arbitration-ID Register	
definition	A-12
example of use during identification phase of initialization	14-20
example of use for arbitration on the AP-bus	7-26
used during the identification phase of initialization	14-6
used for arbitration on the AP-bus	7-24
used for marrying modules by a special agent	14-12
used to marry processor modules using another module	13-9
Asynchronous I/O System	15-2
Attach-Bus Error Type	
definition	12-12
determining which bus failed	13-18
used after a bus switch	13-21
Attach-Bus Command	
definition	A-14
description	13-21
used after a bus switch	13-21
used to issue an error report	12-12

B	phase of initialization	
Bad-Access Reply	example of use during parameterization phase of initialization	
definition	used as the location in an error message	7-16
usage	used for initialization	8-12
BADAC (Bad Access) Signal	used for initialization	3-30
used by cache logic of BXU	used for marking modules for a special event	8-24
used with IAC requests	used in an error message	8-12
used with write requests on the AP-bus		8-39
BE ₅ -BE ₀ (Byte Enable) Signals	used with the attach-bus sequence	
definition	Cache Error Type	3-5
timing		3-5
timing diagram	terminating signal has failed	3-9
used by the BXU and cache	Marked	8-19
used by the byte enable latch		4-5
used by the DRAM controller		4-15
used by the prefetch unit of the BXU	Diagnosis and Control Logic of the BXU	8-35
BERL ₅ -BERL ₀ (Bus Error) Signals	coherency	
definition	configuration and control	7-4
diagnostic tests		12-15
timing		7-32
timing with error message	memory status	12-2
used by the error reporting network	operation	12-1
used during phase one of error reporting sequence		12-3
used during phase two of error reporting sequence		12-10
used in FRC	SRAM operation	11-4
BOUT (Bus Output Control) Signal	SRAM support logic	
definition	system error	7-5
used as a detection mechanism in fault-tolerant systems		11-2
used during initialization		14-3
used in FRC	system operation	11-4
Burst Logic	CACHE signal	
memory interface	used in BXU to M. memory mode	4-3
signal flow	used in a BXU to Processor mode	4-4
Burst Transaction		3-10
Bus Recovery	Cache Error Type	13-18
Bus Switching	cache configuration and control	
communications after a bus switch		13-22
effects on cache accesses		13-25
effects on I/O prefetch accesses		13-26
effects on IAC operations		13-20
effects on memory accesses	terminating the attach-bus	13-25
effects on memory range recognition	new cache timing operation	13-23
overview		13-16
recovery	used in BXU to M. address recognition on the I. bus	13-17
Bus-Arbitration Error Type	used for testing the cache directory	
definition		12-11
determining which bus failed		13-18
Bus-Error-ID Register	testing directory	
definition		A-15
example of use during a processor	definition	
module marriage		14-30
example of use during identification	used as a detection mechanism in fault-tolerant systems	

phase of initialization	14-22,14-23,14-25,14-26
example of use during parameterization phase of initialization	14-19
used as the location in an error message	12-10,12-11,12-12
used during initialization	14-5
used for bus recovery	13-18
used for marrying modules by a special agent	14-12
used in an error message	12-9,12-10,12-11
used to marry processor modules using another module	13-10
used with the attach-bus sequence	13-21
Bus-Parity Error Type	
definition	12-11
determining which bus failed	13-18
Byte Marks	7-13
C	
Cache Directory and Control Logic of the BXU	
coherency	8-13,8-16,8-17
configuration and control	8-16
directory	8-16
fill sequence	8-23
memory structure	8-14
operation	8-13
operation with multiple BXUs	8-17
reaction to bad-access reply	8-24
SRAM Interface	8-18
SRAM support logic	8-19
update policy	8-16
Cache Memory	
attached to the BXU	8-13
retry operation	13-5
CACHE Signal	3-6
used by a BXU in Memory mode	8-25
used by a BXU in Processor mode	8-14
used by the prefetch unit of the BXU	8-33
Cache-Configuration Register	
cache configuration and control	8-16
considerations after a bus switch	13-25
definition	A-17
directory	8-17
effect on replacement algorithm	8-15,8-16
selecting the timing mode	8-10
slow cache timing operation	8-13
timing options	8-40
used for INIT-RAM address recognition on the L-bus	8-10
used for testing the cache directory	8-37
Cache-Test Register	
definition	A-21
testing directory	8-37
CHK_i-CHK₀ (Check) Signals	
definition	7-4
timing	7-32
used as a detection mechanism in fault-tolerant systems	11-2

used in FRC	11-4
CLK2 (Processor Clock) or CLK (Bus Clock)	
AP-bus signal	7-4
CLK2 requirements	3-12
generation	3-12
relationship of CLK2 and CLK	3-6
Coherency	8-13,8-16,8-17
Cold Start	14-9
COM (Communication) Signal	
definition	7-5
used by a memory controller	13-14
used during initialization	14-3
used to indicate an error	12-15
COM Register	
definition	A-23
used by a memory controller interface	13-14
used during the parameterization phase of initialization	14-7,14-8
COM Protocol	14-7
COM-Altered Error Type	
definition	12-12
serial COM protocol	14-8
Commands Listed in Appendix A	
Attach-Bus	A-14
Detach-Bus	A-25
Invalidate-Cache	A-37
Primary-Catastrophe	A-54
Shadow-Catastrophe	A-62
Sync-Refresh	A-64
Terminate-Permanent-Error-Window	A-66
Test-Report	A-69
See also individual commands	
Comparators for I/O System	15-5
Component Error Type	12-12
Component-Specifier Register	
definition	A-24
Configurations With the BXU	8-42
Confinement	
AP-bus	11-1
definition	10-6,11-1
example of operation	11-6
module area	11-3
Confinement-ID Field	12-9
COR (Correct) Signal	13-13
Correctable-ECC Error Type	12-12
CR (Cache Read) Signal	8-18
CW (Cache Write) Signal	8-18
Cycle-In-Progress Signal	
definition	4-3
used by the burst logic	4-3
used by the DRAM arbiter	4-12
used by the DRAM timing and control	4-13,4-14
used by the timing control logic	4-5

D	
Data Transceivers	
I/O interface	5-1
memory interface	4-1
Default Initialization Phase	
definition	14-1
pins sampled	14-3
DEN (Data Enable) Signal	
definition	3-4
timing diagram	3-9
used by a data transceiver	4-1, 5-3
used by the burst logic	4-3
Detach-Bus Error Type	
definition	12-12
determining which bus failed	13-18
used after a bus switch	13-
Detach-Bus Command	
definition	A-25
used after a bus switch	13-18
used to issue an error report	12-12
Detection Mechanism	
definition	10-6, 11-1
FRC	11-3
parity	11-2
signal duplication	11-2
time-outs	11-2
DRAM Address Multiplexer	4-10
DRAM Arbiter	4-12
DRAM Controller	4-10
DRAM Interleaving	4-17
DRAM Refresh Interval Timer	4-11
DRAM Timing and Control	4-12
DRAM Timing Considerations	4-15
DT/R (Data Transmit/Receive)Signal	
definition	3-4
timing diagram	3-7
used by a data transceiver	4-1, 5-3
used by the 82586 interface	5-10
E	
ECC (ECC Error) Signal	13-14
Error Message	12-8
Error Report Log	12-14
Error-Log Register	
defining a permanent error	13-3
defining a transient error	13-1
definition	A-26
error recording	12-14
permanent error decision	13-5
unsafe error decision	13-3
used during phase one of error reporting sequence	12-7
used with the <i>Unsafe-Confinement-Area</i> error type	12-10

Error-Record Register	used for processor module recovery
definition	A-28
error recording	12-14
initialization	14-3
phase two of error reporting sequence	12-7
Error-Reporting-Error Error Type	HILDA (Hold Acknowledge Request) Signal
definition	12-11
determining which bus failed	13-18
permanent error	12-15
Error-Type Field	12-9
F	NO FRC Pair
FAILURE Signal	3-32
Fault-Tolerant I/O System	15-2
FRC (Functional Redundancy Checking)	to an active/passive module
description	11-3
operation and setup	11-6
overview	10-5
physical connection	11-3
FRC Register	description
definition	A-29
FRC operation and setup	11-6
used to marry processor modules using another module	13-8
used to select a BXU	12-16
FRC Splitting	13-26
FRC-Splitting-Control Register	bus recovery ("Identify Device Order" IAC)
definition	A-30
FRC operation and setup	11-6
FRC splitting	13-26
initialization	14-3
used to distinguish between a warm or cold start	14-10
FRF (Force Refresh) Signal	13-15
FTI Register	example of using the "Identify Device Order" IAC
BERL timing	12-3
definition	A-33
example of use during a processor module marriage	14-29
example of use during identification phase of initialization	14-23, 14-26
example of use during parameterization phase of initialization	14-18
used by a memory controller interface	13-13
used during IAC transactions	8-27, 8-28, 8-29, 8-30
used during initialization	14-5
used during phase one of error reporting sequence	12-5
used during the parameterization phase of initialization	14-8
used for bus recovery	13-18
used for processor module recovery	13-12
used in an error type	12-12
used with the attach-bus sequence	13-21
FT2 Register	module recovery using "Register Request Using a Physical Address IAC"
definition	A-35
used during IAC transaction	8-27, 8-28, 8-30
used during the parameterization phase of initialization	14-8
used for bus recovery	13-19

used for processor module recovery	13-12
used with the attach-bus sequence	13-21
H	
HLDA (Hold Acknowledge) Signal	3-17
HLDA (Hold Acknowledge Request) Signal	3-18
HOLD Signal	3-17
HOLDR (Hold Request) Signal	3-18
I	
I/O Address Range	5-3
I/O FRC Pair	15-2
I/O Interface	
to a passive module	9-3
to an active/passive module	9-4
I/O interface to 8-bit and 16-bit Peripherals	5-1
I/O Interface to the 80960MC	5-1
I/O Prefetch Logic	8-32
IAC Generator	
description	9-6
design	9-8
state diagram	9-11
used for marrying a module shadow	14-11
IAC Transactions	
address formats	7-17
bus recovery ("Identify Device Order" IAC)	13-18
bus recovery (Register Request Using a Logical Address" IAC)	13-18
bus recovery ("Register Request Using a Physical Address" IAC)	13-18
bus recovery (IAC Message)	13-18
bus switching	13-16
data flow	7-17
example of using the "Register Request From the L-Bus" IAC	14-19
example of using the "Identify Device Order" IAC	14-20
example of using the "Register Request From the L-Bus" IAC	14-31
example of using the "Register Request Using a Logical Address" IAC	14-24,14-26
example of using the "Register Request Using a Physical Address" IAC	14-22,14-28,14-25
format for the "Identify Device Order" IAC	7-22
format for the "Register Request Using a Physical Address" IAC	7-20
format for the "Register Request Using a Logical Address" IAC	7-19
format for the "Register Request From the L-Bus" IAC	7-21
IAC address recognition (Logical)	8-28
IAC address recognition (Message)	8-29,8-30
IAC address recognition (Physical)	8-28
IAC address recognition (Register)	8-27
marrying processor modules employing "Register Request Using a Logical Address" IAC	13-10
marrying processor modules employing "Register Request Using a Physical Address" IAC	13-7,13-9
message	7-18
module recovery using "Register Request Using a Physical Address IAC	13-11
module recovery using the "Identify Device" IAC	13-11
operation after a bus switch	13-20
retry sequence with IAC message	13-4
ID-Parity Bit	12-10

Identification Phase	14-20
example	14-20
overview	14-5
Identify Device Order	7-22
address format	7-22
definition	7-22
used during initialization	14-6
See also IAC Transactions	definition
INIT-RAM	8-9, 14-5
Initialization	example of use during a processor module manager
for the M82965	14-1
Initialization for 80960MC	3-34
INITID (Initialization Identification) Signal	status after RESET
connection between master and checker	14-32
definition	7-5
used during initialization	14-5
INT ₀ /IAC (Interrupt or Inter-Agent Communication)	used to reset processor mode
Signal	3-7, 3-28
INT ₁ (Interrupt1) Signal	3-37, 3-28
INT ₂ /INTR (Interrupt2 or Interrupt Request)	used during phase one of error reporting sequence
Signal	3-27, 3-28
INT ₃ /INTA (Interrupt3 or Interrupt Acknowledge)	line
Signal	3-28
Interleaving	8-26
Interleaving Cache Memory	8-17
Interrupts	diagnostic testing
definition	3-29
direct interrupt pins	3-26
handling in fault-tolerant systems	15-6
Interrupt Control register	3-26
pins that interface to an interrupt controller	3-26
signals	3-25
synchronization	3-28
timing diagram	3-27
Invalid Bit	12-10
Invalidate-Cache Command definition	A-37
used to synchronize caches	13-9
L-Bus Interface Logic on the BXU functions, list of	8-9
IAC address recognition (logical)	8-28
IAC address recognition (Message)	8-29, 8-30
IAC address recognition (Physical)	8-28
IAC address recognition (Register)	8-27
INIT-RAM address recognition	8-9
L-bus and AP-bus conversions	8-11
memory address recognition	8-9
private memory address recognition	8-10
L-Bus States address (T _a)	3-16, 3-1
data (T _d)	3-16, 3-1
hold (T _h)	3-15
hold request (T _{hr})	3-16
idle (T _i)	3-16, 3-1
recovery (T _r)	3-16, 3-1

LAD (Local Address/Data) lines	3-3
LAD ₁ -LAD ₀	
see SIZE signals	
LBI-Control Register	
arbitration usage	8-25
cache configuration and control	8-16
definition	A-38
example of use during parameterization phase of initialization	14-19
example of use during a processor module marriage	14-29
example of use during identification phase of initialization	14-24, 14-27
interleaving after a bus switch	13-24
state after RESET	14-4
used for INIT-RAM address recognition	
on the L-bus	8-10
used for processor module recovery	13-12
used to select Processor mode	14-3
LERL ₁ -LERL ₀ (Bus Error) signals	
used by the error reporting network	12-1
used during phase one of error reporting sequence	12-5
used during phase two of error reporting sequence	12-11
Line	8-14
Line-Valid	8-17
Local-Bus-Test Register	
definition	A-41
diagnostic testing	8-37
Lock Register	
definition	A-43
memory considerations after a bus switch	13-25
used by a memory controller interface	13-12
used by a memory module	13-12
used during memory operation	8-25
used during RMW operations	8-25
LOCK Signal	
definition	3-5
used during arbitration	3-18
Logical-ID Register	
definition	A-45
example of use during a processor module marriage	14-22
example of use during identification phase of initialization	14-20, 14-22, 14-23, 14-25, 14-26
state after RESET	14-5
used during IAC transactions	7-19, 7-20, 8-28, 8-29, 8-30
used during the identification phase of initialization	14-7
used to identify a BXU	7-2
M	
Mask Register	
definition	A-46
example of use during parameterization phase of initialization	14-19
example of use during a processor module marriage	14-29
used for bus recovery	13-19

Match Register	
BXU operation when Sequence bit set to one	8-13
definition	A-46
example of use during parameterization phase of initialization	14-19
example of use during a processor module of marriage	14-29
interleaving after a bus switch	13-24
used for bus recovery	13-19
used for bus switching	13-17
used to marry processor modules using another module	13-10
used with the attach-bus sequence	13-21
Maxtime Register	
definition	A-49
transient wait period	10-7
MEM/REG (Memory/Register) Signal	
definition	8-26
used by a memory controller	13-14
Memory Address Range	4-3
Memory Controller and BXU	13-13
Memory Interface to the BXU	9-1
Memory Interface to the 80960MC	4-1
Memory Mode	8-2, 8-25
Memory Organization on AP-Bus	7-6
Message Bus	8-27
MODCHK (Module Check) Signal	
definition	7-5
used during initialization	14-3
used in FRC	11-3
Module Confinement Area	11-3
Module Shadowing	
initialization example	14-17
marriage using a special agent	14-11
memory module definition	13-12, 13-14
memory module marriage	13-15
memory module response to error reports	13-12
overview	10-7
processor module definition	13-6
processor module marriage	13-7, 13-15
used for processor module recovery	13-11
Module-Error-ID Register	
definition	A-50
example of use during processor module marriage	14-30
example of use during identification phase of initialization	14-22, 14-23, 14-25, 14-26
example of use during parameterization phase of initialization	14-19
FRC splitting	13-26
permanent error decision	13-5
used as the location in an error message	12-10, 12-11, 12-12, 12-13
used by a memory controller interface	13-15
used during initialization	14-5
used for marrying modules by a special agent	14-12
used in an error message	12-10, 12-11
used to marry processor modules using another module	13-9
NACK (No-Acknowledgement) Reply	

definition	7-16
used during IAC transactions	8-12
used in IAC transactions	8-30
Null Bit	12-9
P	
Packet	7-3
Parameterization Phase	14-7
Partial Write Operations	8-12, 13-14
Partner Communication	10-9
Passive Interface Circuit	15-4
Passive Module	8-44
PBM (Primary Bus Master)	3-20
Permanent Error Decision	13-5
Permanent Errors	13-3
PFETCH (Prefetch) Signal	8-33
Phase One of Error Report	12-5
Phase Two of Error Report	12-7
<i>Physical-ID Register</i>	
definition	A-51
example of use during identification phase of initialization	14-20
state after RESET	14-4
used during IAC transactions	7-21, 8-28
used during the identification phase of initialization	14-5, 14-6
POPQUE Signal	13-22
<i>Prefetch-Control Register</i>	
considerations after a bus switch	13-26
definition	A-52
example of use during a processor module marriage	14-29
example of use during parameterization phase of initialization	14-19
used for bus recovery	13-19
used with the attach-bus sequence	13-21
used with the I/O prefetch unit	8-32
<i>Primary-Catastrophe Error Type</i>	
defining a permanent error	13-3
defining a transient error	13-1
definition	12-11
unsafe error decision	13-3
used during processor module recovery	13-11
<i>Primary-Catastrophe Command</i>	
definition	A-54
when to use	12-11
<i>Primary-Elect Unit</i>	13-9, 14-22
<i>Private Memory Mask Register</i>	
definition	A-55
private memory address recognition	8-10
<i>Private Memory Match Register</i>	
definition	A-55
private memory address recognition	8-10
Processor Mode	8-2
<i>Processor-Message Register</i>	
used during IAC transactions	8-27

used during message IAC requests	8-30, 8-31, 8-32
Processor-Priority Register	
definition	A-56
used during IAC transactions	7-22, 8-27, 7-19
used during message IAC requests	8-30, 8-31, 8-32
write operations after a bus switch	13-20
Q	
QMR (Quad Modular Redundancy)	
initialization example	14-18
overview	10-5
See also Module Shadowing	
QMR Register	
definition	A-58
example of use during a processor module marriage	14-28, 14-31
example of use during identification phase of initialization	14-22, 14-23, 14-24, 14-26
FRC splitting	13-27
used during processor module recovery	13-11
used for marrying modules by a special agent	14-11, 14-12
used for processor module recovery	13-12
used to marry processor modules using another module	13-10, 13-9
used to select a BXU	12-16
QMR I/O	15-6
R	
Read Byte	
definition	7-13
operation	8-11
Read Double-Byte	
definition	7-13
operation	8-11
Read Operation	
retry operation	13-4
sequence during prefetch operation	8-34
sequence on the AP-bus	7-11
Read Operation, timing diagram for the L-bus	3-6
Read-Reply Packet	
operation	7-12
used in RMW operations	7-15
Read Request Packet	
general definition	7-12
operation	7-11
READY Signal	
definition	3-5
timing diagram	3-7
used by the 82586 interface	5-12
used by the M8259A interface	5-7
used by the 82786 interface	5-12, 5-13
used by the BXU and cache	8-19
used by the byte enable latch	4-5
used by the SRAM interface	4-7, 4-9, 4-10
used by the timing control logic	4-5, 5-3

with AP-bus transactions	8-13
Recovery	13-1
Redundancy	13-6
Register Request from the L-Bus	used during IAC transactions
See IAC Transactions	used during message IAC requests
Register Request Using a Logical Address	write operations after a bus switch
See IAC Transactions	
Register Request Using a Physical Address	
See IAC Transactions	
Register, a procedure for initialization	14-2
Registers Listed in Appendix A	overview
Arbitration-ID	A-12
AP-Control	A-8
AP-Mask	A-10
AP-Match	A-10
Bus-Error-ID	A-15
Cache-Configuration	A-17
Cache-Test	A-21
COM	A-23
Component-Specifier	A-24
Error-Log	A-26
Error-Record	A-28
FRC	A-29
FRC-Splitting-Control	A-31
FTI	A-33
FT2	A-35
LBI-Control	A-38
Local-Bus-Test	A-40
Lock	A-42
Logical-ID	A-45
Mask ₀	A-46
Mask ₁	A-46
Mask ₂	A-46
Match ₀	A-46
Match ₁	A-46
Match ₂	A-46
Maxtime	A-49
Module-Error-ID	A-50
Physical-ID	A-51
Prefetch-Control	A-52
Private Memory Mask	A-55
Private Memory Match	A-55
Processor-Priority ₀	A-56
Processor-Priority ₁	A-56
QMR	A-58
Spouse-ID	A-63
System-Bus-ID	A-65
Test-Detection	A-67
Test-Type	A-70
See also individual registers	
Reissue-reply packet	
definition	7-16

used in RMW operations	7-15
Reply Ordering	7-11
Reply Packet	7-7
general definition	7-7
operation on AP-bus	7-11
types of	7-8
Reporting and Recovery	10-7
cycle	10-7
Reporting Network	12-1
Request Packet	7-7
general definition	7-7
operation on AP-bus	7-11
types of	7-7
RESET	14-2
clock phase synchronization	14-1
during default initialization phase	13-28
FRC splitting	7-4
signal on AP-bus	3-29
timing generation for 80960MC	9-7
used by the IAC generator	10-8, 13-3
Retry	7-15
RMW (Read-Modify-Write) Operation	8-11
definition	13-4
use for LOCK signal	7-15
used during retry sequence	7-15
RMW-Read Packet	7-15
RMW-Write Packet	7-4, 7-31
RPYDEF (Reply Deferral) Signal	11-5
definition	3-20
used in FRC	8-15
S	13-3
SBM (Secondary Bus Master)	13-2
Set	12-11
Shadow-Catastrophe Error Type	13-3
defining a permanent error	13-2
defining a transient error	13-3
definition	13-11
unsafe error decision	13-11
used during processor module recovery	A-62
Shadow-Catastrophe Command	12-11
definition	13-9, 14-22
when to use	3-3
Shadow-Elect Unit	5-10
SIZE Signals	4-3
definition	8-36
used by the 82586 interface	7-30
used by the burst logic	7-28
used by the prefetch unit of the BXU	10-9
Slots	
Software Considerations for Reconfiguration	

Source-ID Field	12-10
SPEC ₁ -SPEC ₀ (Operation/Status) Signals	7-8
SPEC ₃ -SPEC ₂ (Cycle Count) Signals	7-8
SPEC ₄ (Multicycle) Signal	7-8
SPEC ₅ (REQUEST/REPLY) Signal	7-8
SPEC ₅ -SPEC ₀ (Specification) Signals	
definition	7-3
used as byte marks	7-13
used during read data transfers	7-1
used in FRC	11-4
used with request and reply packets	7-8
Spouse-ID Register	
definition	A-63
example of use during a processor module marriage	14-31
example of use during identification phase of initialization	14-22, 14-23, 14-25, 14-26
example of use during parameterization phase of initialization	14-20
used for marrying modules by a special agent	14-12
used for processor module recovery	13-11
used to marry processor modules using another module	13-9
SRAM Interface	
as cache with BXU	8-18, 1-19
cache timing diagrams	8-20, 8-21, 8-22
interface logic	4-6
timing considerations	4-6
SSBUSY (Subsystem Busy) Signal	13-22
Start Bit	12-9
Start Command for I/O Prefetch Unit	13-26
Start Channel Command	8-34
Sync-Refresh Error Type	
definition	12-13
memory module shadowing	13-15
Sync-Refresh Command	
definition	A-64
used to issue an error report	12-13
used to synchronize memory	13-15
Synchronizer	15-3
Synchronous I/O System	15-1
System Propagation Delay	7-34
System-Bus-ID Register	
definition	A-65
state after RESET	14-4
used during IAC requests	14-3
used during IAC transactions	8-27, 8-28, 8-29
T	
Tag	8-14, 1-17
Tag-Valid	8-17
Terminate-Permanent-Error-Window Error Type	
definition	12-12
permanent error decision	13-5
Terminate-Permanent-Error-Window Command	
defining a permanent error window	10-9

definition	A-66
permanent error decision	13-5
used to issue an error report	12-12
Test-Detection Register	used to determine warm or cold start
definition	A-67
used during the parameterization phase of initialization	14-8
used for diagnostic testing	12-16
used to generate error reports	12-13
Test-Report Command	definition
definition	A-69
used to select an error type	12-17
used to test retry buffers	13-4
Test-Type Register	used by the DRAM timing control logic
definition	A-70
used to select an error type	12-17
Testing	Way
cache	WP (WRITE) Signals 8-36
cache directory	8-37
L-Bus interface	8-37
Time-Outs on the AP-Bus	with error sequence 11-2
Time-Slide	7-22, 7-24
Timing	Write-Acknowledge Packet
arbitration on the L-bus	used during IAC transactions 3-19
cache read miss	in IAC transactions 8-40, 8-41
I/O prefetch read	in memory transactions 8-41, 8-42
interrupts	used in RMW operations 3-27
of IAC generator	9-7
read operation on the L-bus	general definition 3-6
read operation with the BXU	8-38
RMW operation with the BXU	8-39, 8-40
write operation on the L-bus	3-9
write operation with the BXU	8-39
Timing Logic	
I/O interface	5-3
signal flow	4-5
Transient Errors	13-2
Transient Wait Period	10-9
Type-Parity Bit	12-9
U	
UNC (Uncorrectable ECC) Signal	13-14
Uncorrectable-Array-Error Error Type	12-12
Unsafe Error Decision	13-3
Unsafe-Confinement-Area Error Type	
defining a permanent error	13-3
defining a transient error	13-3
definition	12-10
determining which bus failed	13-18
failures in partial write operations	13-14
unsafe error decision	13-3
Update Policy	8-16

V _{REF} (Voltage Reference)	7-6
definition	14-9
used to determine warm or cold start	8-14
Valid bit	8-14
W _R (Write/Read) Signal	3-4
definition	3-7
timing diagram	5-13
used by the 82786 interface	8-33
used by the BXU during prefetch	4-12
used by the DRAM timing control logic	5-4, 4-5
used by the timing control logic	14-9
Warm Start	8-15
Way	8-19
WD ₇ -WD ₀ (WORD) Signals	7-13
Write Operation	13-4
sequence on the AP-bus	3-10
with retry sequence	8-12
Write Operation, timing diagram on the L-bus	8-31
Write-Acknowledge Packet	8-25
used during IAC transactions	7-15
used in IAC transactions	7-11
used in memory transactions	7-14
used in RMW operations	8-18
Write-Request Packet	
general definition	
operation on AP-bus	
WY ₇ -WY ₀ (WAY) Signals	
Timing Logic	
IO interface	
signal flow	
Transient Errors	
Transient Wait Period	
Type-Parity Bit	
U	
UNC (Uncorrectable ECC) Signal	
Uncorrectable-Army-Error Type	
Unsafe Error Decision	
Unsafe-Confinement-Area Error Type	
defining a permanent error	
defining a transient error	
definition	
determining which bus failed	
failures in parallel write operations	
unsafe error decision	
Update Policy	